TEMA 5: ANÁLISIS SINTÁCTICO ASCENDENTE.

1. INTRODUCCIÓN.

Normalmente vamos a utilizar métodos sin retroceso. Para realizar un análisis sintáctico ascendente de reducción-desplazamiento hay 2 métodos principales:

- Precedencia de operador → Forma más sencilla de aplicar el análisis ascendente.
- LR → Forma más eficiente y general. Realmente es un conjunto de métodos.

Utilizaremos gramáticas de tipo 2 (GCL = Gramáticas de Contexto Libre), sin ciclos y sin reglas λ.

Ejemplo: Dada la gramática:

$S \rightarrow aABe$ (1)

$A \rightarrow Abc \mid b$ (2|3)

$B \rightarrow d$ (4)

Cadena de entrada: $w \equiv abbcde$

La idea del análisis sintáctico es buscar en la cadena de entrada subcadenas que equiparen con la parte derecha de las reglas.

a**bb**c**d**e — Las "b" equiparan con (3) y la "d" con (4). Reducimos la primera "b" con (3).

↓

a**Abcd**e — Reducimos "Abc" por (2).

↓

a**Ad**e — Reducimos "d" por (4).

↓

aABe — Reducimos "aABe" por (1).

↓

S — Axioma de la gramática.

Análisis a derechas: $S \overset{(1)}{\Rightarrow} aABe \overset{(4)}{\Rightarrow} aAde \overset{(2)}{\Rightarrow} aAbcde \overset{(3)}{\Rightarrow} abbcde = w$ (para w)

Elegir siempre el no terminal más a la derecha.

(32)

* **PIVOTE** (de una cadena de tokens): Subcadena que conseguimos equiparar con el lado derecho de una regla para hacer una reducción que forma parte del análisis que me lleva a la solución (reducción que forma parte del árbol sintáctico válido de esa cadena).

Ejemplo: Pivotes del ejemplo anterior:

b
Abc
d
aABe

} Son las cadenas que he ido reduciendo.

· Definición formal.- Un pivote de una forma sentencial λ es una producción (regla) de la forma $A \to B$ asociada a una posición de λ donde puede encontrarse la cadena B y reemplazarse por A para reproducir la anterior forma sentencial derecha dentro de un análisis a derechas de la forma sentencial λ.

* **GRAMÁTICA AMBIGUA:** Son aquellas que producen en general más de un camino para reconocer una misma sentencia de entrada.

Para las gramáticas ambiguas puede haber más de un pivote. Esto es una situación indeseable.

Quiero un único camino, y para ello hay que trabajar con gramáticas no ambiguas. Para cada sentencia correcta habrá un solo árbol sintáctico.

↓

Sólo habrá una subcadena que se pueda reducir dentro de un camino válido (desde la cadena al axioma). En cada paso podré reducir por otras subcadenas pero no me llevarán a ningún lado.

1 CADENA CORRECTA ⇒ 1 ÚNICO ÁRBOL SINTÁCTICO ⇒ EL PIVOTE SIEMPRE EXISTE.

Ejemplo: Gramática ambigua.

Producciones:
$E \to E + E$ 1
$E \to E * E$ 2
$E \to id$ 3

Reduzco siempre el no terminal más a la derecha.

Cadena a analizar: $w \equiv id + id * id$

Derivación 1: $\underset{\text{Reduce +}}{E} \overset{1}{\Rightarrow} E + \underset{\text{Reduce *}}{E} \overset{2}{\Rightarrow} E + E * E \overset{3}{\Rightarrow} E + E * id \overset{3}{\Rightarrow} E + id * id \overset{3}{\Rightarrow} id + id * id$

PIVOTES

Derivación 2: $\underset{\text{Reduce *}}{E} \overset{2}{\Rightarrow} E * \underset{\text{Reduce +}}{E} \overset{3}{\Rightarrow} E * id \overset{1}{\Rightarrow} E + E * id \overset{3}{\Rightarrow} E + id * id \overset{3}{\Rightarrow} id + id * id$

PIVOTES

En este ejemplo tengo dos pivotes distintas en el mismo paso y las dos me llevan a la solución:

$E + E * id$
- Derivación 1.
- Derivación 2.

Por tanto, es una gramática ambigua.

En la derivación 1 damos mayor precedencia (prioridad) al operador *, por ello lo reducimos primero. En cambio, en la derivación 2 tiene mayor precedencia +. Es la 1 la derivación correcta respecto a las reglas de precedencia de los operadores aritméticos.

Hay dos problemas a resolver a la hora de realizar un analizador sintáctico ascendente:

1) Localizar la subcadena que queremos reducir.
2) Seleccionar alguna de las producciones posibles para reducir la subcadena.

A la derecha de los pivotes siempre debo tener símbolos terminales. → Lógico, porque se deriva el no terminal más a la derecha.

Ejemplo: Funcionamiento de un analizador sintáctico ascendente reducción - desplazamiento. No hay retroceso.

Tenemos una pila de trabajo y un buffer de entrada.
El símbolo $ indica fin de cadena y también fondo de la pila.
Las operaciones que permitimos son:

- **Reducción** → Localizar el pivote en la pila de trabajo y sustituirlo por el lado izquierdo de la regla correspondiente (símbolo no terminal).
- **Desplazar** → Coger el siguiente elemento del buffer de entrada y colocarlo en la cima de la pila de trabajo.
- **Aceptar** → Validar la cadena de entrada.
- **Error** → Error en la cadena de entrada.

Producciones de la gramática:

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$

Cadena a analizar → $id_1 * id_2 + id_3$

Con este algoritmo no necesito enumerar las reglas porque en todo momento sabemos qué regla aplicar. Para cada estado de la pila y conociendo el siguiente token, tenemos una acción predeterminada. Esta acción es la única que me validará la cadena. Si no consigo validarla es que la cadena no es correcta.

La acción que tengo que realizar me viene dada por un método que ya veremos.

PILA	ENTRADA	ACCIÓN
$	$id_1 * id_2 + id_3$ $	Desplazar.
id_1 $	* $id_2 + id_3$ $	Reducir F → id
F $	* $id_2 + id_3$ $	Reducir T → F
T $	* $id_2 + id_3$ $	Desplazar ①
* T $	$id_2 + id_3$ $	Desplazar
id_2 * T $	+ id_3 $	Reducir F → id
F * T $	+ id_3 $	Reducir T → T * F ②
T $	+ id_3 $	Reducir E → T ③
E $	+ id_3 $	Desplazar
+ E $	id_3 $	Desplazar
id_3 + E $	$	Reducir F → id
F + E $	$	Reducir T → F
T + E $	$	Reducir E → E + T
E $	$	Aceptar ④

① Al tener T en la cima de la pila y * en la entrada, el método me indica que tengo que desplazar. Quiero dar prioridad al operador *

② Elegimos reducir T → T * F porque queremos dar prioridad al operador * frente al operador +.

③ El algoritmo me dice que tengo que reducir debido a las prioridades de las regias aritméticas.

④ Al tener en la pila el axioma de la gramática y en la entrada el $ (fin de cadena, cadena vacía), podemos aceptar la cadena, pues es una cadena válida.

2. A. SINTÁCTICO ASCENDENTE CON PRECEDENCIA DE OPERADOR.

Con este método vamos a utilizar un tipo especial de gramáticas: **Gramáticas de Precedencia**. Son gramáticas que permiten definir relaciones de precedencia entre los símbolos de la gramática. Estas relaciones nos van a ayudar a encontrar el pivote en cada paso del análisis sintáctico ascendente.

Las características que debe cumplir una gramática para ser de precedencia son:

- Ser una gramática propia (sin ciclos ni reglas λ).
- No aparecen 2 símbolos no terminales adyacentes en la parte derecha de las reglas ($A \rightarrow aBCd$ no vale).
- Existe como máximo una relación de precedencia entre cualquier par de símbolos de la gramática.

Las relaciones de precedencia que vamos a utilizar son:

1) $a \lessdot b$: a tiene menos precedencia que b.
2) $a \doteq b$: a tiene igual precedencia que b.
3) $a \gtrdot b$: a tiene más precedencia que b.

Esta precedencia se refiere a la precedencia aritmética de los símbolos matemáticos.

Estos métodos surgieron para analizar expresiones aritméticas y se pueden trasladar a cualquier sentencia de un lenguaje de programación.

Para calcular las relaciones de precedencia para todos los símbdos de la gramática se usa la tabla de relaciones de precedencia. Por ejemplo, una tabla podría ser:

R.P.	a	b	c
a		$\lessdot$	$\doteq$
b	$\lessdot$	$\lessdot$	$\lessdot$
c	$\lessdot$	$\gtrdot$	$\doteq$

En cada casilla puede haber una única relación o ninguna. Un hueco en la casilla significa que esa relación no puede producirse, es decir, no pueden aparecer esos dos símbolos seguidos en una cadena válida. Si ocurriera eso se produciría un error.

En una tabla puede aparecer $a \lessdot b$ y $b \lessdot a$. Esto es correcto, pues las relaciones de precedencia no son simétricas. El orden de los símbolos es fundamental porque se refiere al orden en el que aparecen los símbolos en la cadena de entrada.

También es correcto que existan jerarquías circulares: $a \lessdot b \lessdot c \lessdot a$.

LAS RELACIONES DE PRECEDENCIA NO SON SIMÉTRICAS:
$a \lessdot b$ NO IMPLICA $b \gtrdot a$

Dada una cadena de entrada, ¿cómo sé cuál es el pivote?

* CÁLCULO DEL PIVOTE:

Dada una forma sentencial $w \equiv \alpha\beta w$, β es el pivote si y sólo si se cumplen las siguientes condiciones:

1) $\lessdot$, $\doteq$ entre símbolos de α.
2) $\lessdot$ entre el último símbolo de α y el primero de β.
3) $\gtrdot$ entre el último símbolo de β y el primero de w.
4) $\doteq$ entre los símbolos de β.

Una manera mecánica de calcularlo es: voy buscando (de izquierda a derecha) el primer $\gtrdot$ y cuando lo encuentro retrocedo hasta que encuentro un $\lessdot$. De esta manera siempre tengo perfectamente determinado cuál es el pivote.

PIVOTE = Primera subcadena encerrada entre $\lessdot$ y $\gtrdot$.

Ejemplo: Cadena de entrada → $w \equiv u\ v\ w\ x\ y\ z$ $\in (T\ U\ N)$ → Forma sentencial.

$\lessdot\ \doteq\ \lessdot\ \doteq\ \gtrdot$

Pivote = xy

Las precedencias las obtengo de la tabla de precedencias, que ya veremos cómo construirla.

2.1. GRAMÁTICA DE PRECEDENCIA DE OPERADOR.

Como ya hemos visto, este método requiere el uso de gramáticas de precedencia, las cuales deben cumplir:

- Ser una gramática sin ciclos ni reglas λ.
- No aparecer 2 símbolos no terminales adyacentes en la parte derecha de las reglas.
- Existir como máximo una relación de precedencia entre cualquier par de símbolos de la gramática.

Esto nos lleva a tener que aumentar las condiciones que deben cumplir las reglas de la gramática.

> Las reglas de las gramáticas de precedencia serán de la forma:
>
> $$A \rightarrow \beta_0 a_1 \beta_1 a_2 \beta_2 \dots a_n \beta_n \quad , \quad a_i \in T, \ \beta_i \in (N \cup \lambda)$$

Ejemplo: Producciones de la gramática:

$E \rightarrow E \ op \ E$

$E \rightarrow id$

$op \rightarrow +$

$op \rightarrow *$

No cumple las condiciones exigidas porque hay 2 (en este caso 3) metanociones (símbolos no terminales) juntos.

Para resolver el problema se sustituye el símbolo op por sus posibles valores:

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$

} Gramática equivalente a la anterior y que cumple las restricciones de una Gramática de Precedencia de Operador.

2.2. TABLA DE RELACIONES DE PRECEDENCIA.

Una vez que tenemos la gramática con precedencia de operador, hay que construir la tabla de relaciones de precedencia. Para ello hay 4 métodos:

1.- Método intuitivo (a ojo).
2.- Método basado en las derivaciones que se pueden ejecutar en la gramática.
3.- Método basado en la definición de los conjuntos Head y Tail.
4.- Método basado en el cálculo de matrices. (No lo estudiamos).

(1) MÉTODO INTUITIVO.

Vamos a estudiar este método intuitivo (a ojo) mediante la gramática del ejemplo anterior:

$$\left.\begin{array}{l} E \to E + E \\ E \to E * E \\ E \to id \end{array}\right\}$$

Símbolos ⇒ id + * $

Final de cadena.

T.R.	id	+	*	$
id		⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	

1) Si aparece + y luego *, siempre hay que reducir primero el *. Por ello, si recibo un + tengo que esperar.
La relación de precedencia es: + ⋖ * , a + b * c.
↳ Primero se hace b*c.

2) Si aparece * y luego +, hay que reducir * primero.
La relación de precedencia es: * ⋗ + , a * b + c
↳ Primero se hace a*b.

3) Si aparece + y luego +, matemáticamente da igual qué suma hacer primero, pero aquí tenemos que ser deterministas. Para ello se toma la asociatividad por la izquierda.
La relación de precedencia es: + ⋗ + , a + b + c.
↳ Primero se hace a+b.

4) Si aparece * y luego *, matemáticamente da igual qué multiplicación hacer primero, pero aquí tenemos que ser deterministas. Para ello se toma la asociatividad por la izquierda.
La relación de precedencia es: * ⋗ * , a * b * c
↳ Primero se hace a*b.

5) El símbolo de exponente (**) es lógico que sea asociativo por la derecha, pues es menos costoso computacionalmente hablando.

6) El símbolo id debe ser lo primero que reducimos, por ello su precedencia debe ser mayor con respecto a la suma, producto y $.

7) El símbolo $ (fin de cadena) nunca lo queremos reducir, por lo que su precedencia debe ser menor con respecto a la suma, producto y el id.

8) La posición correspondiente a (id, id) se deja en blanco. Nunca podremos tener dos identificadores seguidos porque son lo primero que reducimos.

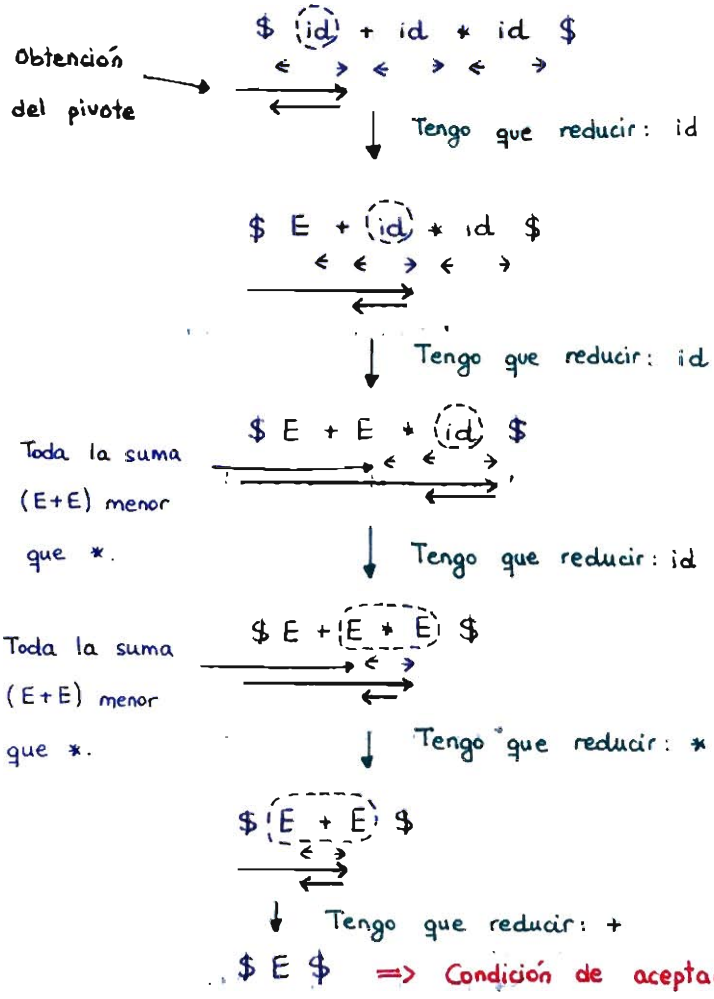
9) La posición correspondiente a ($, $) se deja en blanco también, pues nunca podremos tener dos $ juntos.

Ejemplo: Vamos a recorrer la cadena:

$w = \$\ id\ +\ id\ *\ id\ \$$

$< \ > \ < \ > \ < \ >$

$E \to E + E$
$E \to E * E$
$E \to id$



> Al definir las relaciones de precedencia, cuando tengo dos operadores con la misma precedencia matemática, los defino asociativos por la izquierda. Así hago determinista el compilador.

La tabla de relaciones de precedencia viene determinada por la gramática. Si la gramática es de precedencia de operador entonces obtengo una única relación para cada casilla. El problema es cuando obtengo varias relaciones para alguna casilla. Hay 2 posibilidades para solucionarlo:

- Cambiar la gramática → Puede resultar costoso y complicado.
- Eliminar una de las relaciones → Surge el problema de que ya no se reconoce lo mismo que reconocía la gramática original.

② MÉTODO BASADO EN LAS DERIVACIONES QUE SE PUEDEN EJECUTAR.

Este método es automático, pero es difícil de implementar.

- Notación que vamos a utilizar:

  a, b ≡ Símbolo terminal (T).
  A, B ≡ Símbolo no terminal (N).
  α, β, γ, δ ≡ (N ∪ λ).

- Método: Sea una gramática de precedencia de operador. Se dan las siguientes reglas:

Reduce los 2 símbolos terminales a la vez.

1) $a \doteq b$ si ∃ una regla del tipo $A \to \alpha a \beta b \gamma$.
2) $a \lessdot b$ si ∃ una regla del tipo $A \to \alpha a B \beta$ y $B \overset{+}{\Rightarrow} \gamma b \delta$.
3) $a \gtrdot b$ si ∃ una regla del tipo $A \to \alpha B b \beta$ y $B \overset{+}{\Rightarrow} \gamma a \delta$.

1) a y b en la misma regla.
2) Una regla contiene "a" y luego un N que deriva finalmente a algo que contiene "b".
3) Una regla contiene "b" y antes un N que deriva finalmente a algo que contiene "a".

Ej. 1) $a \doteq b$ si ∃ $A \to \alpha a \beta b \gamma$

$S \to A a b C$
$A \to d$
$C \to g$
} Gramática

$\$ \; d \; a \; b \; g \; \$$ (A sobre d, C sobre g, $\doteq$ entre a y b)

$\alpha = A$
$\beta = \lambda$
$\gamma = C$

Ej. 2) $a \lessdot b$ si $\exists\, A \to \alpha a B \beta$ y $B \overset{+}{\Rightarrow} \gamma b \delta$

$S \to aAB$
$A \to b$
$B \to c$
} Gramática

$ a b c $

Ej. 3) $a \doteq b$ si $\exists\, A \to \alpha b B \beta$ y $B \overset{+}{\Rightarrow} \gamma a \delta$

$S \to AbB$
$A \to a$
$B \to d$
} Gramática

$ a b $

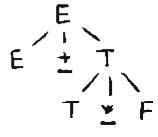
Ejemplo:

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$

T.R.	+	*	(	)	id	$
+		⋖ ②				
*	⋗ ③					
(				≐ ①		
)						
id						
$						

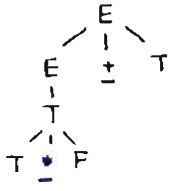
1) $F \to (E)$ $\quad A \to \alpha a \beta b \gamma$ $\quad \alpha = \lambda,\ \beta = E,\ \gamma = \lambda$ $\quad (\doteq)$

2) $E \to E + T$
$T \to T * F$
$\quad A \to \alpha a B \beta$ y $B \overset{+}{\Rightarrow} \gamma b \delta$ $\quad \alpha = E,\ \beta = \lambda$ $\quad + \lessdot *$



3) $E \to E + T$
$E \to T$
$T \to T * F$
$\quad A \to \alpha B b \beta$ y $B \overset{+}{\Rightarrow} \gamma a \delta$ $\quad \alpha = \lambda,\ \beta = T$ $\quad * \gtrdot +$

$E \to T \to T * F$



Ejemplo:

$E \to E + E$
$E \to E * E$

① - Aplicando el punto 2 obtengo:

$E \to E + E$ $\quad A \to \alpha a B \beta \quad B \to \gamma b \delta$ $\quad + \lessdot *$
$E \to E * E$

② - Aplicando el punto 3 obtengo:

$E \to E + E$ $\quad A \to \alpha B b \beta \quad B \to \gamma a \delta$ $\quad * \gtrdot +$ ←
$E \to E * E$

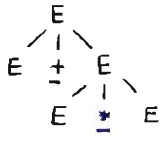
③ - Aplicando el punto 2 obtengo:

$E \to E * E$ $\quad A \to \alpha a B \beta \quad B \to \gamma b \delta$ $\quad * \lessdot +$ ←
$E \to E + E$

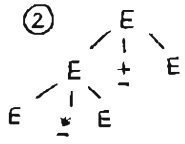
↑ He cambiado el orden de las reglas de derivación (el orden no importa).

→ Entonces tendría una casilla con 2 relaciones de precedencia entre dos terminales. Tengo 2 opciones para eliminar este problema:

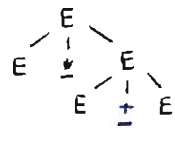
- Cambiar la gramática para que refleje las precedencias.
- Eliminar una de las relaciones de precedencia. El problema es que entonces el analizador sintáctico ya no reconoce el mismo lenguaje que la gramática y esto puede ser peligroso.

① 

$* \gtrdot +$

② 

$+ \lessdot *$

③ 

$* \lessdot +$

(Más prioridad el de más abajo).

③ MÉTODO BASADO EN LOS CONJUNTOS HEAD Y TAIL.

Este método consta de 3 pasos:

1) $\forall A \in N$ calcular los conjuntos:

$Head(A) = \{ a \in T / A \overset{*}{\Rightarrow} Ba \_\_, B \in N \text{ ó } \lambda$
$Tail(A) = \{ a \in T / A \overset{*}{\Rightarrow} \_\_ aB, B \in N \text{ ó } \lambda$

- $Head(A) = \{ a / A \overset{+}{\Rightarrow} \alpha a \beta \;\; , \;\; \alpha \in (N \cup \lambda), \beta \in (N \cup T)^* \}$
- $Tail(A) = \{ a / A \overset{+}{\Rightarrow} \beta a \gamma \;\; , \;\; \beta \in (N \cup T)^*, \gamma \in (N \cup \lambda) \}$

2) $\forall$ producción $A \rightarrow x_1 x_2 \dots x_n$ repetir:

Para $i = 1..n-1$:

A cada producción aplicar todas las reglas posibles, no sólo una. Ej: si tengo TNT aplicar la 2ª y 3ª

- Si $(x_i \in T)$ y $(x_{i+1} \in T) \Rightarrow x_i \doteq x_{i+1}$
- Si $(x_i \in T)$ y $(x_{i+1} \in N)$ y $(x_{i+2} \in T) \Rightarrow x_i \doteq x_{i+2}$ $(i = 1..n-2)$
- Si $(x_i \in T)$ y $(x_{i+1} \in N)$, $\forall a \in Head(x_{i+1}) \Rightarrow x_i \lessdot a$
- Si $(x_i \in N)$ y $(x_{i+1} \in T)$, $\forall a \in Tail(x_i) \Rightarrow a \gtrdot x_{i+1}$

3) $\forall a \in Head(S) \Rightarrow \$ \lessdot a$ $\qquad$ $S \equiv$ Axioma.
$\forall b \in Tail(S) \Rightarrow b \gtrdot \$$

Ejemplo:

$$\left.\begin{array}{l} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid id \end{array}\right\} \text{ Gramática}$$

Paso 1.

$N \in \{ E, T, F \}$
$T \in \{ +, *, (,), id \}$

$Head(E) = \{ +, *, (, id \}$ $\qquad$ $Tail(E) = \{ +, *,), id \}$

$Head(T) = \{ *, (, id \}$ $\qquad$ $Tail(T) = \{ *,), id \}$

$Head(F) = \{ (, id \}$ $\qquad$ $Tail(F) = \{), id \}$

Paso 2:

* $E \rightarrow E + T$
  $N\ T\ N$
  $x_1\ x_2\ x_3$

  $i = 1 \rightarrow Tail(E) \gtrdot + \Rightarrow \{+, *,), id\} \gtrdot +$
  $i = 2 \rightarrow + \lessdot Head(T) \Rightarrow + \lessdot \{*, (, id\}$

* $E \rightarrow T$ (No hago nada)

* $T \rightarrow T * F$
  $N\ T\ N$
  $x_1\ x_2\ x_3$

  $i = 1 \rightarrow Tail(T) \gtrdot * \Rightarrow \{*,), id\} \gtrdot *$
  $i = 2 \rightarrow * \lessdot Head(F) \Rightarrow * \lessdot \{(, id\}$

* $T \rightarrow F$ (No hago nada)

* $F \rightarrow (E)$
  $T\ N\ T$
  $x_1\ x_2\ x_3$

  $i = 1 \rightarrow (\ \doteq\ )$
  $(\lessdot Head(E) \Rightarrow (\lessdot \{+, *, (, id)\}$
  $i = 2 \rightarrow Tail(E) \gtrdot) \Rightarrow \{+, *,), id\} \gtrdot)$

* $F \rightarrow id$ (No hago nada)

Paso 3:

$\$ \lessdot Head(E) \Rightarrow \$ \lessdot \{+, *, (, id\}$
$Tail(E) \gtrdot \$ \Rightarrow \{+, *,), id\} \gtrdot \$$

T.R.	+	*	(	)	id.	$
+	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
*	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
)	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
id	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

2.3. ALGORITMO DE RECONOCIMIENTO (PRECEDENCIA DE OPERADOR).

Una vez construida la tabla con las relaciones de precedencia, se usa un algoritmo de reconocimiento.

* GRAMÁTICA ESQUELÉTICA.

Sea $G \equiv (N, T, P, S)$ una gramática de precedencia de operador. Se define $G_S \equiv (\{S\}, T, P', S)$ como la gramática esquelética correspondiente a G. La gramática esquelética tiene un único símbolo no terminal, que coincide con su axioma. La lista de producciones P' es:

$$P' : (\{S \to x_1 \dots x_n\} \;/\; \exists\, A \to y_1 \dots y_n \in P)$$

donde

$$\begin{cases} x_i = y_i & \text{si } y_i \in T \\ x_i = S & \text{si } y_i \in N \end{cases}$$

$L(G) \neq L(G_S)$

Las gramáticas reconocen lenguajes diferentes.

Ejemplo:

Gramática original:

$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to id$

Gramática esquelética:

$B \to B + B$
~~$B \to B$~~
$B \to B * B$
~~$B \to B$~~
$B \to (B)$
$B \to id$

Estas dos producciones no sirven, las podemos eliminar de nuestra gramática esquelética.

La gramática esquelética se define después de la creación de la tabla de relaciones de precedencia. Esta gramática la utilizaremos para el reconocimiento de la cadena de entrada (algoritmo). Todos los pasos anteriores hay que hacerlos con la gramática original.

ALGORITMO DE ANÁLISIS SINTÁCTICO POR PRECEDENCIA DE OPERADOR

- Entrada. Una cadena de entrada w y una tabla de relaciones de precedencia de operador.
- Salida. Si w es sintácticamente correcta, se habrá construido en la pila implícitamente su árbol de análisis (se habría podido construir explícitamente si se hubiera necesitado); en caso contrario, la salida será una indicación de error sintáctico. → Parse a derechas.
- Método. Inicialmente la pila contiene el símbolo de fondo de pila, $, y el "buffer" de entrada contiene la cadena y su delimitador, w$. El algoritmo de análisis es el siguiente:

1)

2) *punt_ent* apunta al primer símbolo de w$;

3) repeat ← $ está en la cabeza de la pila.

3.1) if el contenido de la pila es $S y *punt_ent* apunta a $

then return (* **aceptar** *)

3.2) else

begin

sea a el símbolo terminal de cima de pila y b el apuntado por *punt_ent*;

3.2.1) if a⋖b o a≐b

then begin (* **desplazar** b *)

meter b en la pila;

avanzar *punt_ent* al siguiente símbolo de la entrada; ← token

end;

3.2.2) else if a⋗b

then (* **reducir** por S → α *)

repeat (* sacar de la pila el consecuente de la regla *)

sacar el elemento de la cima de la pila

until el terminal de la cima de la pila esté relacionado por ⋖

con el terminal más recientemente extraído;

sea $\alpha = a_p \ldots a_j$ la cadena de elementos sacados de la pila, con a_p el más recientemente extraído, y sea $S \rightarrow \alpha \in P$; } Se puede omitir.

meter S en la pila (* meter el antecedente *)

3.2.3) else error () (* **error** *)

↳ Error si no hay relación de precedencia.

end

4) El parse vendrá dado por los números de las reglas cuyas partes derechas se hayan sacado de la pila en el paso 3.2.2.

Ejemplo:

Gramática original:

$E \to E + T$

$E \to T$

$T \to T * F$

$T \to F$

$F \to (E)$

$F \to id$

Gramática esquelética

1 $B \to B + B$

2 $B \to B * B$

3 $B \to (B)$

4 $B \to id$

T.R	+	*	(	)	id	$
+	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
*	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
)	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
id	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

← Calculada anteriormente.

Cadena de entrada:
w ≡ id * (id + id) $

PILA	PUNT-ENT	REL. PREC.	PRODUCCIÓN
$	id	$\lessdot$	Desplazar.
$ ~~id~~	*	$\gtrdot$	B → id . Reducir 4.
$ B	*	$\lessdot$	Desplazar.
$ *	(	$\lessdot$	Desplazar.
$ * (	id	$\lessdot$	Desplazar.
$ * (~~id~~	+	$\gtrdot$	B → id. Reducir 4.
$ * (	+	$\lessdot$	Desplazar.
$ * (+	id	$\lessdot$	Desplazar.
$ * (+ ~~id~~	)	$\gtrdot$	B → id. Reducir 4.
$ * (~~+~~	)	$\gtrdot$	B → B + B. Reducir 1.
$ * (	)	$\doteq$	Desplazar.
$ * ~~( ) ~~	$	$\gtrdot$	B → (B). Reducir 3.
$ ~~*~~	$	$\gtrdot$	B → B * B. Reducir 2.
$	$		Aceptar.

se puede omitir

Parse generado = 4 4 4 1 3 2.

3. A. SINTÁCTICO ASCENDENTE LR.

Tenemos un grupo de métodos que denominaremos de forma genérica **métodos LR(K)**, donde las iniciales tienen el siguiente significado:

- L ≡ Left: La cadena de entrada se lee (analiza) de izquierda a derecha.
- R ≡ Right: Se realiza un análisis a derechas, es decir, un análisis ascendente.
- K: Indica el número de símbolos de la cadena de entrada (número de tokens) que debemos conocer anticipadamente para tomar decisiones (para hacer el análisis).
  K suele valer 0 ó 1 ⟹ LR(0) ó LR(1).

De manera general, cuando hablemos de LR(1) escribiremos sólo LR.

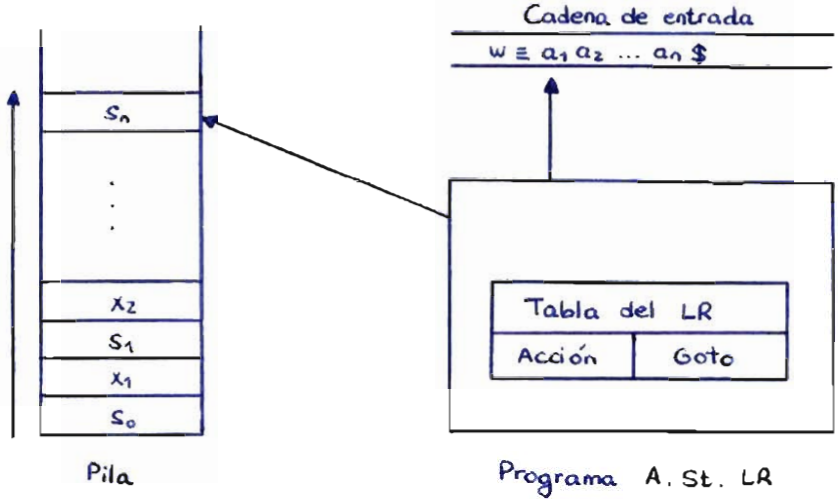
Hay diferentes métodos LR ≈ LR(1):

* LR Simple (SLR) → Es el método más sencillo. (El que vamos a ver).
* LR Canónico.
* Look Ahead LR (LALR) → LR con análisis anticipado. Es el que implementa la herramienta Yacc.

Estos métodos LR son los más generales para realizar un análisis sintáctico ascendente de reducción - desplazamiento, más generales que el m precedencia de operador, pues aceptan gramáticas que generan cualquier construcción posible de un lenguaje de programación común. Las gramáticas con precedencia de operador son un subconjunto de las gramáticas que reconoce el análisis LR. Los LR admiten más gramáticas de tipo 2. Además, estos métodos son tan eficientes como el resto de ascendentes y los errores se detectan en el primer instante que se pueden detectar. Como inconveniente, es mucho más complejo elaborar la información necesaria para aplicar el método LR.

Los tres métodos antes nombrados se diferencian en las tablas que hay que crear para poder aplicar el algoritmo.

ESQUEMA GENERAL DEL A. SINTÁCTICO LR:



Acción ⇒ Cuándo actuamos.
Goto ⇒ Dónde ir después de una reducción.

La pila siempre va a tener alternancia de estado y símbolo. En la cabeza siempre debe haber un estado.

Los estados tratan de resumir lo que ha pasado por la pila, incluyen el conocimiento de qué hemos hecho hasta el momento.

Operaciones permitidas:

Estado inicial → ($s_0 X_1 s_1 X_2 s_2 \dots s_{n-1} X_{n-1} s_n$, $a_i \dots a_n \$$)
pila entrada

· **Desplazamiento:**

Si Acción $(s_n, a_i) = d_j$ (desplazar e ir al estado j) ⇒

($s_0 X_1 s_1 X_2 s_2 \dots s_{n-1} X_{n-1} s_n a_i s_j$, $a_{i+1} \dots a_n \$$)

· **Reducción:**

Si Acción $(s_n, a_i) = r_k$ (reducir por la regla k) ⇒

($s_0 X_1 s_1 X_2 s_2 \dots s_{n-r} A s_e$, $a_i \dots a_n \$$)

Los últimos elems. coinciden con la regla (el consecuente). Lo sustituyo por el antecedente.

regla k ≡ $A \to \alpha$; $|\alpha| = r$ ← nº de elems. del consecuente.

s_e = GOTO (s_{n-r} , A)
↳ estado al que vamos.

· **Error:**

Si Acción $(s_n, a_i) = \phi$

· **Aceptar:**

Indica cuándo termina con éxito el análisis sintáctico.

Ejemplo:

* Gramática:

$E \to E + T$	r1
$E \to T$	r2
$T \to T * F$	r3
$T \to F$	r4
$F \to (E)$	r5
$F \to id$	r6

$N = \{ E, T, F \}$

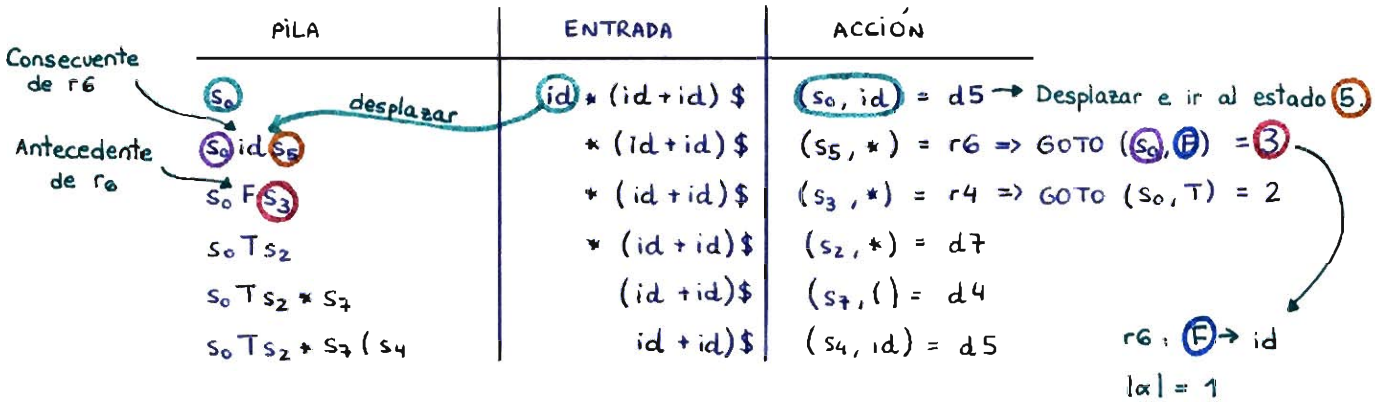
$T = \{ id, +, *, (,) \}$

* Cadena de entrada: $w \equiv id * (id + id)$

* Tabla del LR:

ESTADO	ACCIÓN						GOTO		
	id	+	*	(	)	$	E	T	F
s_0	d5			d4			1	2	3
s_1		d6				Acep.			
s_2		r2	d7		r2	r2			
s_3		r4	r4		r4	r4			
s_4	d5			d4			8	2	3
s_5		r6	r6		r6	r6			
s_6	d5			d4				9	3
s_7	d5			d4					10
s_8		d6			d11				
s_9		r1	d7		r1	r1			
s_{10}		r3	r3		r3	r3			
s_{11}		r5	r5		r5	r5			

* Algoritmo:



PILA	ENTRADA	ACCIÓN
$s_0 T s_2 * s_7 (s_4 id s_5$	+ id) $	$(s_5, +) = r6 \Rightarrow GOTO(s_4, F) = 3$
$s_0 T s_2 * s_7 (s_4 F s_3$	+ id) $	$(s_3, +) = r4 \Rightarrow GOTO(s_4, T) = 2$
$s_0 T s_2 * s_7 (s_4 T s_2$	+ id) $	$(s_2, +) = r2 \Rightarrow GOTO(s_4, E) = 8$
$s_0 T s_2 * s_7 (s_4 E s_8$	+ id) $	$(s_8, +) = d6$
$s_0 T s_2 * s_7 (s_4 E s_8 + s_6$	id) $	$(s_6, id) = d5$
$s_0 T s_2 * s_7 (s_4 E s_8 + s_6 id s_5$	) $	$(s_5,)) = r6 \Rightarrow GOTO(s_6, F) = 3$
$s_0 T s_2 * s_7 (s_4 E s_8 + s_6 F s_3$	) $	$(s_3,)) = r4 \Rightarrow GOTO(s_6, T) = 9$
$s_0 T s_2 * s_7 (s_4 E s_8 + s_6 T s_9$ (Consecuente de r1)	) $	$(s_9,)) = r1 \Rightarrow GOTO(s_4, E) = 8$
$s_0 T s_2 * s_7 (s_4 E s_8$ (Antecedente de r1)	) $	$(s_8,)) = d11$
$s_0 T s_2 * s_7 (s_4 E s_8) s_{11}$	$	$(s_{11}, \$) = r5 \Rightarrow GOTO(s_7, F) = 10$
$s_0 T s_2 * s_7 F s_{10}$	$	$(s_{10}, \$) = r3 \Rightarrow GOTO(s_0, T) = 2$
$s_0 T s_2$	$	$(s_2, \$) = r2 \Rightarrow GOTO(s_0, E) = 1$
$s_0 E s_1$	$	$(s_1, \$) =$ Aceptar.

Análisis → 6 4 6 4 2 6 4 1 5 3 2 (Reglas aplicadas)

3.1. CONSTRUCCIÓN TABLAS LR (ACCIÓN + GOTO).

Vamos a ver el **método SLR** (LR Simple). Es el método más sencillo pero el menos general, pues es el que admite menor número de gramáticas.

Las gramáticas que permiten la construcción de un método SLR son las **gramáticas SLR**. No se sabe si una gramática es o no SLR hasta que hayamos construido la tabla.

El método se basa en:

1.- Prefijo viable.
2.- Item.
3.- Gramática aumentada.
4.- Colección canónica.
5.- First y follow.

3.1.1. PREFIJO VIABLE.

Los prefijos viables son todas las formas sentenciales a derechas que pueden aparecer en la pila de trabajo de un analizador sintáctico ascendente por reducción - desplazamiento.

> Un prefijo viable es un prefijo de una forma sentencial derecha que no continúa más allá del extremo derecho del pivote de dicha forma sentencial derecha.

Ejemplo: Dada la forma sentencial → a A b c D E , siendo D el pivote.
Pivotes viables:

- a
- a A
- a A b
- a A b c
- a A b c D

} Todos los prefijos que van hasta incluir el pivote.

3.1.2. ITEM.

> Un item es una producción de la gramática a la que coloco un punto en alguna posición del lado derecho. El punto determina hasta dónde he analizado, es decir, en qué estado del análisis estamos para poder reducir esa regla.

Ejemplo: Dada la producción: $A \rightarrow XY$
Los items asociados son:

- $A \rightarrow .XY$ ≈ No tengo nada para poder reducir por esta regla. Indica deseos.
- $A \rightarrow X.Y$ ≈ Indica cuánto he avanzado para reducir esa regla. Tengo X, pero me falta Y.
- $A \rightarrow XY.$ ≈ Tengo todos los elementos para poder reducir por esta regla y así obtener A.

Los items nos van a permitir encontrar todos los prefijos viables.

3.1.3. GRAMÁTICA AUMENTADA.

Dada una gramática $G(N, T, P, S)$.
Se define G' como la gramática aumentada tal que:

$$G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$$

Definir la gramática aumentada sirve para ver cuándo se acepta una cadena. Una cadena es aceptada al aplicar la regla $S' \to S$. Así, en el algoritmo, en lugar de poner "Aceptar" pondríamos r_i siendo $r_i : S' \to S$.

Además, la gramática aumentada sirve para evitar conflictos cuando el axioma aparece en el lado derecho de una regla.

3.1.4. COLECCIÓN CANÓNICA.

Es el pilar fundamental para la construcción de la tabla LR. Es un conjunto de conjuntos de items.

Para construir la colección canónica hay que partir de la gramática aumentada y es necesario conocer dos funciones: cierre y GOTO.

* **CIERRE de un conjunto de items:**

Representa todas las posibilidades que tengo abiertas en un momento dado.

Sea I un conjunto de items de G, se define el conjunto de items Cierre (I) de la siguiente manera:

a) $\forall a \in I \Rightarrow a \in$ Cierre (I). Es decir, todo elemento del conjunto de items se añade al cierre ($I \subseteq$ Cierre (I)).

b) Si $A \to \alpha . B\beta \in$ Cierre (I) y $B \to \gamma \in G \Rightarrow$
$B \to .\gamma \in$ Cierre (I) (siendo B un símbolo no terminal).

Repetir hasta que Cierre (I) no cambie.

Ejemplo:

Dada la gramática G:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Calculamos la gramática aumentada:

$E' \rightarrow E$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Conjunto de items → $I = \{ E' \rightarrow . E \}$

¿Cuál es el cierre?

Cierre (I) = $\{ E' \rightarrow .E$ ①, $E \rightarrow . E + T$ ②, $E \rightarrow .T$ ③, $T \rightarrow . T * F$ ④, $T \rightarrow . F$ ⑤,

$F \rightarrow . (E)$ ⑥, $F \rightarrow . id \}$ ⑦

① Este item lo incluyo en el cierre porque pertenece al conjunto inicial de items (I) (apartado a).

② Como E es un símbolo no terminal, busco las reglas que tengan la forma $E \rightarrow$ "algo" y las incluyo poniendo un punto delante.

③ Misma razón que ②.

④ Como T es un símbolo no terminal, busco las reglas que tengan la forma $T \rightarrow$ "algo" y las incluyo poniendo un punto delante.

⑤ Misma razón que ④.

⑥ Como F es un símbolo no terminal, busco las reglas que tengan la forma $F \rightarrow$ "algo" y las incluyo poniendo un punto delante.

⑦ Misma razón que ⑥.

(*) Ya he terminado porque lo que tengo en el cierre no cambia, pues en ⑥ y ⑦ lo que tengo a continuación del punto son símbolos terminales.

(44)

* **GOTO:**

La función GOTO(I, X) se define para un conjunto de ítems I y un símbolo X de la gramática ($X \in N \cup T$).

Se define GOTO(I, X) como el cierre del conjunto formado por todos los ítems de la forma:

$$A \to \alpha X . \beta \quad \text{donde} \quad A \to \alpha . X \beta \in I$$

Ejemplo:

Sea $I = \{ E' \to .E \,,\; E \to E. + T \}$

Sea la gramática:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid id$

$E' \to E$ → (Gramática aumentada).

Vamos a calcular **GOTO(I, +)**:

Para ello tengo que mirar en I (conjunto de ítems) y localizar las reglas en las que aparezca .+ ⟹ Selecciono $E \to E . + T$.

Ahora cambio el orden del punto ⟹ $E \to E + . T$

Finalmente, calculo el cierre:

$$GOTO(I, +) = Cierre(\{E \to E + . T\}) = \{ E \to E + . T \,,\; T \to . T * F \,,\; T \to . F \,,\; F \to . (E) \,,\; F \to . id \}$$

El conjunto de ítems va a determinar el estado del analizador. La función GOTO me dice que, estando en el estado I y apareciendo el símbolo X, voy a pasar a alguno de los estados (ítems) que obtengo como resultado de la función GOTO.

* COLECCIÓN CANÓNICA:

La colección canónica es un cjto de cjtos de ítems. Se calcula a partir de la gramática aumentada mediante el siguiente algoritmo.

Dada una gramática G tipo 2 y calculada la gramática aumentada G'.

1. Definir el primer conjunto de items: I_0 = Cierre ({ S' → .S })

2. Inicializamos la colección canónica: C = { I_0 }

3. Contador j = 1.

4. Repetir (hasta que C no cambie) ⊛
   para cada $I_i \in C$ y $\forall X \in (NUT)$ → Cada símbolo gramatical X.
   (Conjunto de items en C)
   calcular GOTO (I_i, X) = I_j
   si ($I_j \neq \phi$) ∧ ($I_j \notin C$) entonces
   añadir I_j a C.
   j = j + 1.

⊛ La colección canónica es un conjunto de I_i, es decir, un conjunto de conjuntos de items.

El algoritmo lo que hace es:

GOTO (I_0, x_1) = I_1
GOTO (I_0, x_2) = I_2
⋮
GOTO (I_1, x_1) = I_j
⋮

A continuación veremos un ejemplo.

Ejemplo:

Vamos a calcular la colección canónica de la gramática aumentada G:

G: $E' \to E$ → Producción de la gramática aumentada.

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid id$

Símbolos: $N = \{ E, T, F \}$ → No se pone E' (son los símbolos de la gramática original).

$T = \{ +, *, (,), id \}$

$X = \{ E, T, F, +, *, (,), id \}$

1. I_0 = Cierre $(E' \to .E)$ = $\{ E' \to .E \ , \ E \to .E + T \ , \ E \to .T \ , \ T \to .T * F \ , \ T \to .F \ , \ F \to .(E) \ , \ F \to .id \}$ I_0

2. C = I_0 (Inicializar).

3. j = 1.

No busco nada. No hay nada después del punto.

No hay reglas que empiecen por +

* 4. I_1 = GOTO (I_0, E) = cierre $(\{ E' \to E. \ , \ E \to E. + T \})$ = $\{ E' \to E. \ , \ E \to E. + T \}$ I_1

Meto reglas con .E

I_2 = GOTO (I_0, T) = cierre $(\{ E \to T. \ , \ T \to T. * F \})$ = $\{ E \to T. \ , \ T \to T. * F \}$ I_2

I_3 = GOTO (I_0, F) = cierre $(\{ T \to F. \})$ = $\{ T \to F. \}$ I_3

I_4 = GOTO $(I_0, +)$ = cierre $(\phi) = \phi$

I_4 = GOTO $(I_0, *)$ = cierre $(\phi) = \phi$

I_4 = GOTO $(I_0, ()$ = cierre $(\{ F \to (.E) \})$ =

= $\{ F \to (.E) \ , \ E \to .E + T \ , \ E \to .T \ , \ T \to .T * F \ , \ T \to .F \ , \ F \to .(E) \ , \ F \to .id \}$ I_4

I_5 = GOTO $(I_0,))$ = cierre $(\phi) = \phi$

$I_5 = GOTO(I_0, id) = cierre(\{F \to id.\}) = \boxed{\{F \to id.\}}$ I_5

* $I_6 = GOTO(I_1, E) = cierre\ \phi = \phi$

$I_6 = GOTO(I_1, T) = cierre\ \phi = \phi$

$I_6 = GOTO(I_1, F) = cierre\ \phi = \phi$

$I_6 = GOTO(I_1, +) = cierre(\{E \to E+.T\}) = \boxed{\{E \to E+.T,\ T \to .T*F,\ T \to .F,\ F \to .(E),\ F \to .id\}}$ I_6

$I_7 = GOTO(I_1, *) = cierre(\phi) = \phi$

$I_7 = GOTO(I_1, () = cierre(\phi) = \phi$

$I_7 = GOTO(I_1,)) = cierre(\phi) = \phi$

$I_7 = GOTO(I_1, id) = cierre(\phi) = \phi$

* $I_7 = GOTO(I_2, E) = cierre(\phi) = \phi$

$I_7 = GOTO(I_2, T) = cierre(\phi) = \phi$

$I_7 = GOTO(I_2, F) = cierre(\phi) = \phi$

$I_7 = GOTO(I_2, +) = cierre(\phi) = \phi$

$I_7 = GOTO(I_2, *) = cierre(\{T \to T*.F\}) = \boxed{\{T \to T*.F,\ F \to .(E),\ F \to .id\}}$ I_7

$I_8 = GOTO(I_2, () = cierre(\phi) = \phi$

$I_8 = GOTO(I_2,)) = cierre(\phi) = \phi$

$I_8 = GOTO(I_2, id) = cierre(\phi) = \phi$

Una vez sacados todos los I_i que salen de I_0, hay que hacer lo mismo para cada I_i sacado anteriormente.

⇓

Se genera GOTO en amplitud: se genera todo lo de I_0 antes de seguir con I_1.

* $I_8 = \text{GOTO}(I_3, E) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, T) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, F) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, +) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, *) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, ()) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3,)) = \text{cierre}(\phi) = \phi$

$I_8 = \text{GOTO}(I_3, id) = \text{cierre}(\phi) = \phi$

* $I_8 = \text{GOTO}(I_4, E) = \text{cierre}(\{ F \to (E.), E \to E.+T \}) =$

$= \boxed{\{ F \to (E.), E \to E.+T \}}$ I_8

$I_9 = \text{GOTO}(I_4, T) = \text{cierre}(\{ E \to T., T \to T.*F \}) =$

$= \boxed{\{ E \to T., T \to T.*F \}}$ Igual que I_2 => No lo añadimos a la colección canónica C.

$I_9 = \text{GOTO}(I_4, F) = \text{cierre}(\{ T \to F. \}) = \boxed{\{ T \to F. \}}$ Igual que I_3 => No añadir a C.

$I_9 = \text{GOTO}(I_4, +) = \text{cierre}(\phi) = \phi$

$I_9 = \text{GOTO}(I_4, *) = \text{cierre}(\phi) = \phi$

$I_9 = \text{GOTO}(I_4, () = \text{cierre}(\{ F \to (.E) \}) =$

$\boxed{\{ F \to (.E),\ E \to .E+T,\ E \to .T,\ T \to .T*F,\ T \to .F,\ F \to .(E),\ F \to .id \}}$

Igual que I_4 => No añadir a C.

$I_9 = \text{GOTO}(I_4,)) = \text{cierre}(\phi) = \phi$

$I_9 = GOTO(I_4, id) = cierre(\{F \to id.\}) = \boxed{\{F \to id.\}}$ Igual que I_5 ⇒ No añadir a C.

* $I_9 = GOTO(I_5, E) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, T) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, F) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, +) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, *) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, () = cierre(\phi) = \phi$

$I_9 = GOTO(I_5,)) = cierre(\phi) = \phi$

$I_9 = GOTO(I_5, id) = cierre(\phi) = \phi$

* $I_9 = GOTO(I_6, E) = cierre(\phi) = \phi$

$I_9 = GOTO(I_6, T) = cierre(\{E \to E+T.\ ,\ T \to T.*F\}) =$

$= \boxed{\{E \to E+T.\ ,\ T \to T.*F\}}$ I_9

$I_{10} = GOTO(I_6, F) = cierre(\{T \to F.\}) = \boxed{\{T \to F.\}}$ Igual que I_3 ⇒ No añadir a C.

$I_{10} = GOTO(I_6, +) = cierre(\phi) = \phi$

$I_{10} = GOTO(I_6, *) = cierre(\phi) = \phi$

$I_{10} = GOTO(I_6, () = cierre(\{F \to (.E)\}) = \boxed{\{F \to (.E),\ E \to .E+T,\ E \to .T,\ T \to .T*F,\ T \to .F,\ F \to .(E),\ F \to .id\}}$

Igual que I_4 ⇒ No añadir a C.

$I_{10} = GOTO(I_6,)) = cierre(\phi) = \phi$

$I_{10} = GOTO(I_6, id) = cierre(\{F \to id.\}) = \boxed{\{F \to id.\}}$ Igual que I_5 ⇒ No añadir a C.

* $I_{10} = GOTO(I_7, E) = cierre(\phi) = \phi$

$I_{10} = GOTO(I_7, T) = cierre(\phi) = \phi$

$I_{10} = GOTO(I_7, F) = cierre(\{T \to T * F.\}) = \boxed{\{T \to T * F.\}}$ I_{10}

$I_{11} = GOTO(I_7, +) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_7, *) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_7, () = cierre(\{F \to (.E)\}) = \boxed{\{F \to (.E),\ E \to .E+T,\ E \to .T,\ T \to .T*F,\ T \to .F,\ F \to .(E),\ F \to id\}}$

Igual que I_4
↳ No añadir a C.

$I_{11} = GOTO(I_7,)) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_7, id) = cierre(\{F \to id.\}) = \boxed{\{F \to id.\}}$ Igual que I_5 ⇒ No añadir a C.

* $I_{11} = GOTO(I_8, E) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_8, T) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_8, F) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_8, +) = cierre(\{E \to E+.T\}) = \boxed{\{E \to E+.T,\ T \to .T*F,\ T \to .F,\ F \to .(E),\ F \to .id\}}$

Igual que I_6 ⇒ No añadir a C

$I_{11} = GOTO(I_8, *) = cierre(\phi) = \phi$

$I_{11} = GOTO(I_8, () = cierre(\phi) = \phi$

$I_{11} = GOTO(I_8,)) = cierre(\{F \to (E).\}) = \boxed{\{F \to (E).\}}$ I_{11}

$I_{12} = GOTO(I_8, id) = cierre(\phi) = \phi$

* I_{12} = GOTO (I_9, E) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9, T) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9, F) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9, +) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9, *) = cierre ({ T → T *. F }) = { T → T *. F , F → .(E), F → .id }

Igual que I_7 ⇒ No añadir a C.

I_{12} = GOTO (I_9, ( ) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9,)) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_9, id) = cierre (ϕ) = ϕ

* I_{12} = GOTO (I_{10}, E) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10}, T) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10}, F) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10}, +) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10}, *) = cierre (ϕ) = ϕ

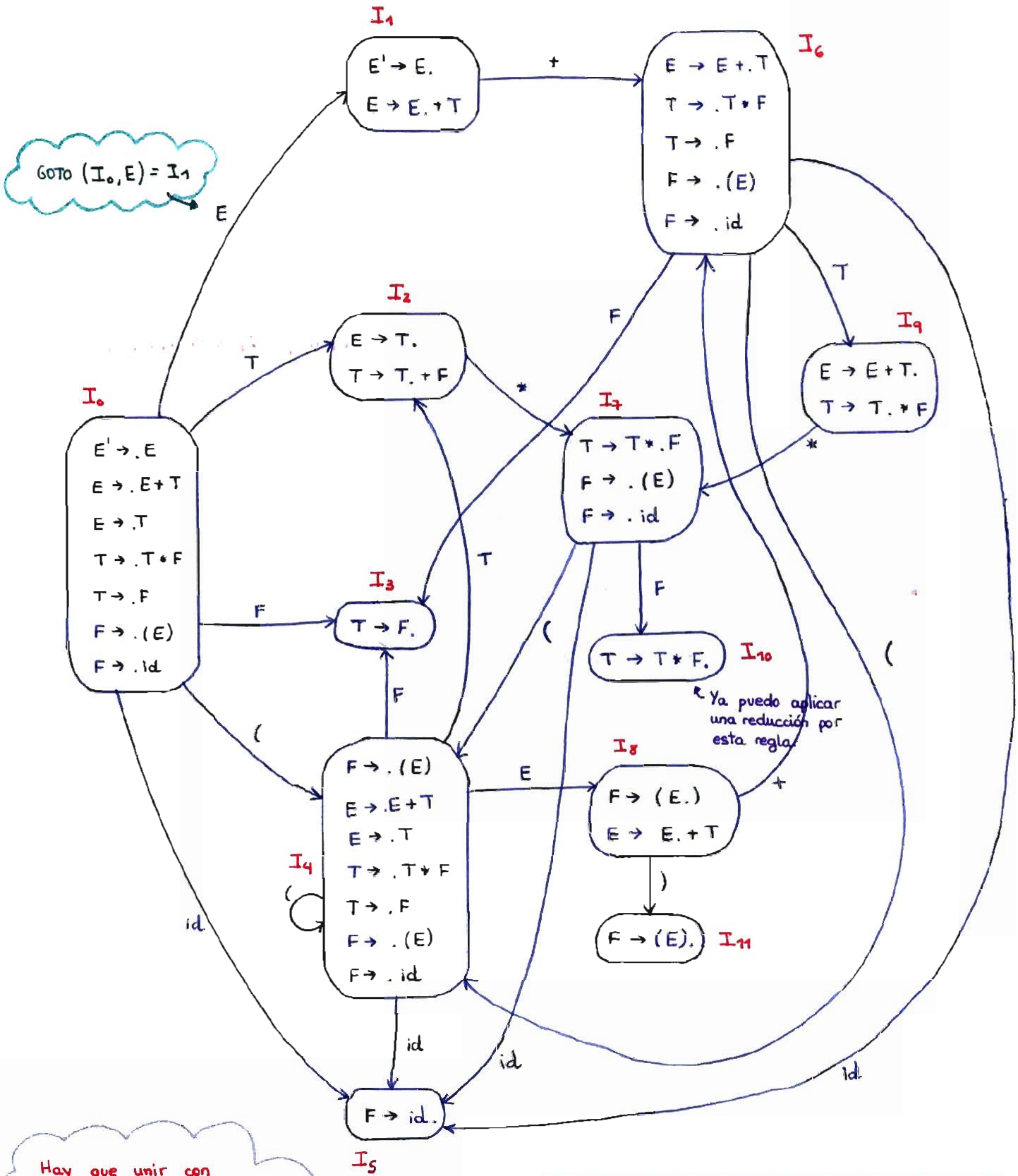
I_{12} = GOTO (I_{10}, ( ) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10},)) = cierre (ϕ) = ϕ

I_{12} = GOTO (I_{10}, id) = cierre (ϕ) = ϕ

* I_{12} = GOTO (I_{11}, E) = cierre (ϕ) = ϕ = GOTO (I_{11}, T) = GOTO (I_{11}, F) = GOTO (I_{11}, +) = GOTO (I_{11}, *) = GOTO (I_{11}, ( ) = GOTO (I_{11},)) = GOTO (I_{11}, id)

Representación gráfica de la colección canónica.



Hay que unir con flechas los I_i siempre: cuando me salía un I_i nuevo y cuando me salía un I_i repetido y ya no lo incorporaba a C.

Las reglas tipo $A \to \lambda$ no dan problemas en los LR:

$S \to aAb$
$A \to \lambda$

Se pone $A \to .$ no $A \to .\lambda$

3.1.5. FIRST Y FOLLOW.

FIRST

Si α es cualquier cadena de símbolos gramaticales, se considera *FIRST(α)* como el conjunto de terminales que encabezan las cadenas derivadas de α. Si $\alpha \overset{*}{\rightarrow} \lambda$, entonces λ también está en *FIRST(α)*.

Para calcular *FIRST(X)* para algún símbolo *X* de la gramática, se aplican las siguientes reglas hasta que no se pueda añadir nada nuevo al conjunto *FIRST*:

1. Si *X* es terminal, entonces *FIRST(X)* es *{X}*.
2. Si *X* es no terminal y existe la producción $X \rightarrow \lambda$, entonces añadir λ a *FIRST(X)*.
3. Si *X* es no terminal y $X \rightarrow Y_1 Y_2 \ldots Y_k$ es una producción entonces, para todo *i* (con *i* variando desde 1 hasta *k*) tal que $Y_1, Y_2, \ldots, Y_{i-1}$ sean todos no terminales y $FIRST(Y_1)$, $FIRST(Y_2)$, ..., $FIRST(Y_{i-1})$ contengan todos λ, se añaden todos los símbolos no nulos de $FIRST(Y_i)$ a *FIRST(X)*. Finalmente, si λ está en $FIRST(Y_j)$ para $j = 1, 2, \ldots, k$ (o sea, en todos), entonces se añade λ a *FIRST(X)*.

Dicho de otra forma, lo anterior significa que todos los elementos de $FIRST(Y_i)$, excepto λ, pertenecen también a *FIRST(X)*. Si Y_i no deriva λ, entonces ya ha terminado el cálculo de *FIRST(X)*, pero en caso contrario, es decir, si $Y_i \overset{*}{\rightarrow} \lambda$, entonces todos los elementos de $FIRST(Y_2)$ excepto λ pertenecen también a *FIRST(X)*, y así sucesivamente. Finalmente, si todos los Y_i derivan λ, entonces λ se añade a *FIRST(X)*.

First (X) nos proporciona los símbolos que encabezan cadenas que se derivan de X.

$$X \rightarrow abCd \Rightarrow a \in First(X)$$

Ejemplo:

X → AB
A → 1 | λ
B → 2 | λ

First (X) = { 1 , 2 , λ }

Porque todos los Y_i derivan λ.

Como A puede ser λ, la cadena podría empezar por los mismos símbolos que las derivaciones de B.

Ejemplo:

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

No hay casos λ.

Símbolos terminales: First (+) = {+} First ()) = {)}
First (*) = {*} First (id) = {id}
First (() = {(}

Símbolos no terminales: First (E) = {(, id}
First (T) = {(, id}
First (F) = {(, id}

→ Hemos empezado con First (E) pero no podemos sacar nada, pues ninguna de sus reglas deriva en un símbolo terminal. Luego empezamos con First (T) y ocurre lo mismo. Al hacer entonces First (F) vemos que sí tiene dos símbolos terminales posibles:

$F \to (E)$
$F \to id$

Toda cadena que se derive de estas dos reglas empezará por (o id.

⇓

First (F) = {(, id}

→ Como tenemos la regla $T \to F$ entonces First (T) = First (F) = {(, id}
Como tenemos la regla $E \to T$ entonces First (E) = First (T) = {(, id}

→ página 38.

Diferencias First - Head:

* First se define $\forall x \in N \cup T$; Head para los N.
* First tiene en cuenta λ; Head no.

FOLLOW

Se define *FOLLOW(A)*, para el no terminal A, como el conjunto de terminales a que pueden aparecer inmediatamente a la derecha de A en alguna forma sentencial, es decir, el conjunto de terminales a tal que haya una derivación de la forma $S \rightarrow \alpha A a \beta$ para algún α y β. Si A puede ser el símbolo de más a la derecha en alguna forma sentencial, entonces $ está en *FOLLOW(A)*.

Para calcular *FOLLOW(A)* para un símbolo no terminal A, se aplican las siguientes reglas hasta que no se pueda añadir nada más al conjunto *FOLLOW*.

1. $ está en *FOLLOW(S)*, siendo S el axioma de G.
2. Si existe una producción $A \rightarrow \alpha B \beta$, entonces todo lo que esté en *FIRST(β)*, excepto λ, está en *FOLLOW(B)*.
3. Si existe la producción $A \rightarrow \alpha B \beta$ y *FIRST(β)* contiene λ (es decir, $\beta \rightarrow \lambda$), o bien si existe una producción $A \rightarrow \alpha B$, entonces todo lo que esté en *FOLLOW(A)* está en *FOLLOW(B)*.

Ejemplo:

E → E + T | T
T → T * F | F
F → (E) | id

Follow (F) = { *, $, +,) }
↑ Por T → F (3)
Follow (T) = { *, $, +,) }
Por E → T (3)
Follow (E) = { $, +,) }
↳ First ( ) (2)
↳ First (+) (2)
↳ Porque E es el axioma (1)

(Reglas aplicadas)

3.1.6. Construcción de la tabla de análisis de un SLR

ENTRADA:

Una gramática aumentada G' y Follow(A) $\forall A \in N$

SALIDA:

Las funciones *ACCION* y *GOTO* de la tabla de un analizador sintáctico SLR para G'

MÉTODO:

1. Construir la colección canónica de ítems LR(0) para G':

   $C = \{I_0, I_1, ..., I_n\}$

   La fila i-ésima se calcula a partir del estado i-ésimo.

2. El estado i es el que se obtiene a partir de I_i. Las acciones para el estado i se determinan de la siguiente manera:

   a) Si $[A \rightarrow \alpha . a \beta]$ está en I_i y $goto(Ii,a)=Ij$ entonces $ACCION[i,a]$ = ***"desplazar j"***. (1) En este caso, a debe ser un terminal, y j es el estado que se apila tras el símbolo desplazado.

   b) Si $[A \rightarrow \alpha .]$ está en I_i entonces $ACCION[i,a]$ = ***"reducir por $A \rightarrow \alpha$"***, y esto para todo terminal a perteneciente a *FOLLOW(A)*. En este caso, A es cualquier no terminal excepto S'.

   c) Si $[S' \rightarrow S .]$ está en I_i entonces $ACCION[i,\$]$ = ***"aceptar"***.

   Si estas reglas generan conflicto en alguna casilla, se dice que la gramática no es SLR(1). En tal caso, no es posible construir un analizador sintáctico mediante este algoritmo.

3. Las transiciones *GOTO* del estado i se construyen para todos los no terminales A mediante la regla: si $goto(I_i,A)=I_j$ entonces $GOTO[i,A] = j$. (2)

4. Todas las casillas no definidas por las reglas 2 y 3 quedan en blanco y corresponden a los casos de ***"error"***.

5. El estado inicial del analizador es el construido a partir del conjunto de ítems que contiene a $[S' \rightarrow . S]$.

dx = Desplazar al estado "x".

ry = Reducir por la regla "y" de la gramática.

(1) Voy de I_x a I_y con el símbolo terminal "a" => En el estado "x", columna "a" pongo "dy" (Desplazar al estado "y").

(2) GOTO → Voy de I_x a I_y con el símbolo no terminal "A" => En el estado "x" (fila "x"), columna "A" pongo "y".

Ejemplo: Construir la tabla LR de la siguiente gramática G:

$E \rightarrow E + T \mid T$ r1 | r2
$T \rightarrow T * F \mid F$ r3 | r4
$F \rightarrow (E) \mid id$ r5 | r6

Para resolverlo necesito.

* La colección canónica → Su representación gráfica está en la página 48.

* Follow (A) $\forall A \in N$ → Ya los calculamos anteriormente:

Follow (E) = { $, +,) }
Follow (T) = { $, +,), * }
Follow (F) = { $, +,), * }

ESTADO	ACCIÓN						GOTO		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Acep.			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

3.1.7. CONFLICTOS EN LA TABLA LR.

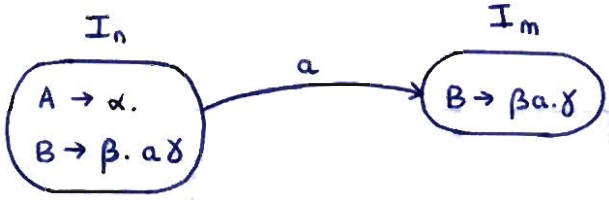
Si una de las casillas de la tabla LR tiene más de una acción, entonces existe un conflicto ⇒ La gramática no vale para este tipo de análisis, no es una gramática SLR(1).

Hay 2 tipos de conflicto:

* Reducción - Desplazamiento.
* Reducción - Reducción.

No se da el conflicto Desplazamiento - Desplazamiento porque la gramática no es ambigua.

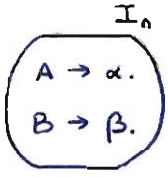
- Conflicto REDUCCIÓN - DESPLAZAMIENTO:



Si $a \in$ Follow(A) ⇒ Conflicto Reducción - Desplazamiento.

Tengo 2 posibles acciones → Acción(n, a) = { Reducir por $A \to \alpha$ / Desplazar a m.

- Conflicto REDUCCIÓN - REDUCCIÓN:

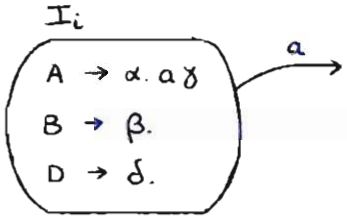


Si tienen algún elemento en común.

Si Follow(A) ∩ Follow(B) $\neq \emptyset$ ⇒ Conflicto Reducción - Reducción.

Tengo 2 posibles reglas para reducir. No sé cuál elegir.

Para comprobar que no hay conflicto:



Si: $a \notin Follow(B)$, $a \notin Follow(D)$, $Follow(B) \cap Follow(D) = \phi$ } ∄ conflicto.

AFD reconocedor de prefijos variables = Colección canónica.

EJERCICIO: FEBRERO '98.

1.- AFD reconocedor de prefijos variables.
2.- Si el estado de la pila es 0S1, explicar todas las posibles acciones del analizador.
3.- Representar las pilas intermedias entre 0S1 y 0S1A3 (cadena de entrada + coste).
4.- Sustituir $B \to c$ por $B \to a$. Analizar si la nueva gramática es SLR(1).

$G \equiv$ $S \to SA \mid b$
$A \to Aa \mid Ab \mid B$
$B \to dBc \mid c$

1) Lo primero es calcular la gramática aumentada:

$G' \equiv$ $S' \to S$, $S \to SA$, $S \to b$, $A \to Aa$, $A \to Ab$, $A \to B$, $B \to dBc$, $B \to c$

Ahora calculamos la colección canónica:

1. I_0 = cierre $(\{S' \to .S\}) = \boxed{\{S' \to .S,\ S \to .SA,\ S \to .b\}}$ I_0
2. $C = I_0$ (Inicializar).
3. $j = 1$.

$N = \{S, A, B\}$
$T = \{a, b, c, d\}$
$X = \{S, A, B, a, b, c, d\}$

* 4. $I_1 = GOTO(I_0, S) = cierre(\{ S' \to S. \ , \ S \to S.A \}) =$

$$= \boxed{\{ S' \to S. \ , \ S \to S.A \ , \ A \to .Aa \ , \ A \to .Ab \ , \ A \to .B \ , \ B \to .dBc \ , \ B \to .c \}} \ I_1$$

$I_2 = GOTO(I_0, A) = \phi = GOTO(I_0, B) = GOTO(I_0, a)$

$I_2 = GOTO(I_0, b) = cierre(\{ S \to b. \}) = \boxed{\{ S \to b. \}} \ I_2$

$I_3 = GOTO(I_0, c) = \phi = GOTO(I_0, d)$

* $I_3 = GOTO(I_1, S) = \phi$

$I_3 = GOTO(I_1, A) = cierre(\{ S \to SA. \ , \ A \to A.a \ , \ A \to A.b \}) =$

$$= \boxed{\{ S \to SA. \ , \ A \to A.a \ , \ A \to A.b \}} \ I_3$$

$I_4 = GOTO(I_1, B) = cierre(\{ A \to B. \}) = \boxed{\{ A \to B. \}} \ I_4$

$I_5 = GOTO(I_1, a) = \phi = GOTO(I_1, b)$

$I_5 = GOTO(I_1, c) = cierre(\{ B \to c. \}) = \boxed{\{ B \to c. \}} \ I_5$

$I_6 = GOTO(I_1, d) = cierre(\{ B \to d.Bc \}) = \boxed{\{ B \to d.Bc \ , \ B \to .dBc \ , \ B \to .c \}} \ I_6$

* $I_7 = GOTO(I_2, S) = \phi = GOTO(I_2, A) = GOTO(I_2, B) = GOTO(I_2, a) =$
$= GOTO(I_2, b) = GOTO(I_2, c) = GOTO(I_2, d)$

* $I_7 = GOTO(I_3, S) = \phi = GOTO(I_3, A) = GOTO(I_3, B)$.

$I_7 = GOTO(I_3, a) = cierre(\{ A \to Aa. \}) = \boxed{\{ A \to Aa. \}} \ I_7$

$I_8 = GOTO(I_3, b) = cierre(\{ A \to Ab. \}) = \boxed{\{ A \to Ab. \}} \ I_8$

$I_9 = GOTO(I_3, c) = \phi = GOTO(I_3, d)$

* $I_9 = GOTO(I_4, X) = \phi \quad \forall X$

* $I_9 = GOTO(I_5, X) = \phi \quad \forall X$

* $I_9 = GOTO(I_6, S) = \phi = GOTO(I_6, A)$

$I_9 = GOTO(I_6, B) = cierre(\{B \to dB.c\}) = \boxed{\{B \to dB.c\}}\ I_9$

$I_{10} = GOTO(I_6, a) = \phi = GOTO(I_6, b)$

$I_{10} = GOTO(I_6, c) = cierre(\{B \to c.\}) = \boxed{\{B \to c.\}}$ Igual que I_5 => No añadir a C.

$I_{10} = GOTO(I_6, d) = cierre(\{B \to d.Bc\}) = \boxed{\{B \to d.Bc,\ B \to .dBc,\ B \to .c\}}$ Igual que I_6 => No añadir a C.

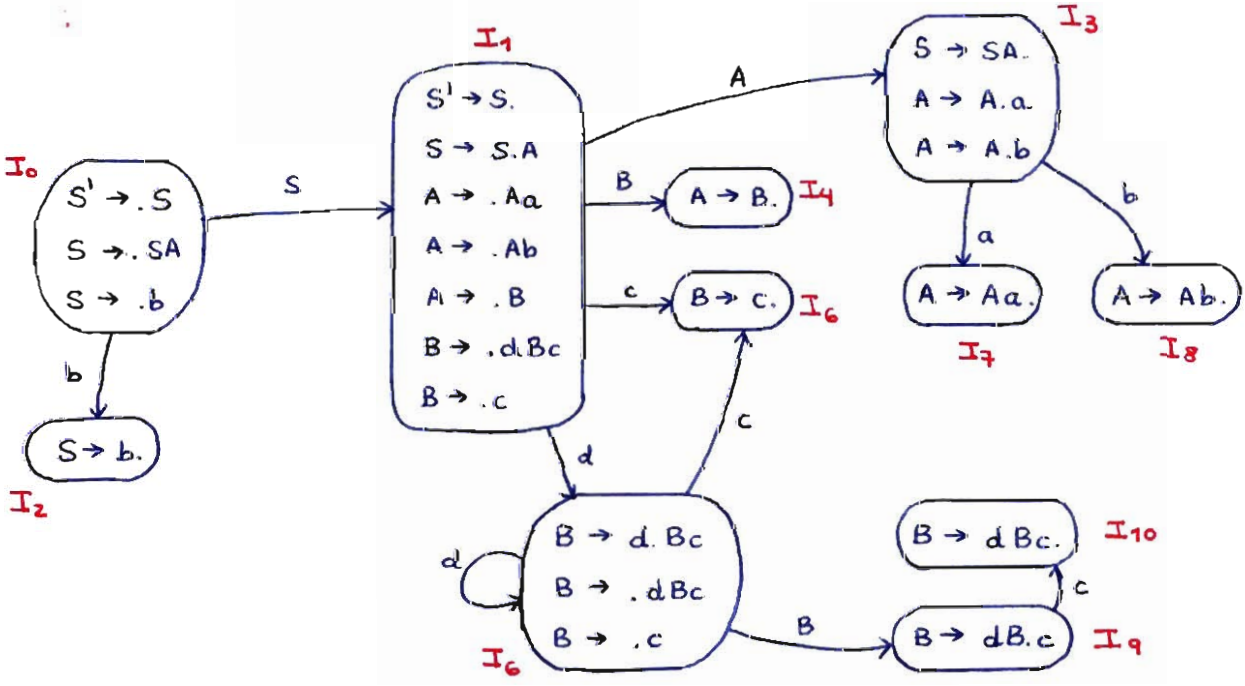
* $I_{10} = GOTO(I_7, X) = \phi = GOTO(I_8, X) \quad \forall X$

* $I_{10} = GOTO(I_9, S) = \phi = GOTO(I_9, A) = GOTO(I_9, B) = GOTO(I_9, a) = GOTO(I_9, b)$

$I_{10} = GOTO(I_9, c) = cierre(\{B \to dBc.\}) = \boxed{\{B \to dBc.\}}\ I_{10}$

$I_{11} = GOTO(I_9, d) = \phi$

* $I_{11} = GOTO(I_{10}, X) = \phi \quad \forall X$



Ejercicio:

A → D ; S
D → D ; D
D → id integer
D → λ
E → id
E → num
S → S ; S
S → id := E
S → λ
A' → A → Gramática extendida.

A es el axioma → Es el único símbolo no terminal que no está en el lado derecho de una producción.

Calculamos la colección canónica:

1. I_0 = cierre ({A' → . A}) = { A' → . A , A → . D ; S , D → . D ; D , D → . id integer , D → . } I_0

2. C = I_0 (Inicializar)

3. j = 1
N = { A, D, S, E }
T = { ; , id , integer , num , := }
X = { A , D , S , E , ; , id , integer , num , := }

* 4. I_1 = GOTO (I_0, A) = cierre ({ A' → A. }) = { A' → A. } I_1

I_2 = GOTO (I_0, D) = cierre ({ A → D. ; S , D → D. ; D }) = { A → D. ; S , D → D. ; D } I_2

I_3 = GOTO (I_0, S) = ∅ = GOTO (I_0, E) = GOTO (I_0, ;)

I_3 = GOTO (I_0, id) = cierre ({ D → id. integer }) = {D → id. integer } I_3

I_4 = GOTO (I_0, integer) = ∅ = GOTO (I_0, num) = GOTO (I_0, :=)

* I_4 = GOTO (I_1, X) = ∅ ∀X

* $I_4 = \text{GOTO}(I_2, A) = \phi = \text{GOTO}(I_2, D) = \text{GOTO}(I_2, S) = \text{GOTO}(I_2, E)$

$I_4 = \text{GOTO}(I_2, ;) = \text{cierre}(\{ A \to D;.S,\ D \to D;.D \}) =$

$$= \boxed{\begin{array}{l}\{ A \to D;.S,\ D \to D;.D,\ S \to .S;S, \\ S \to .id := E,\ S \to .,\ D \to .D;D, \\ D \to .id\ integer,\ D \to . \}\end{array}} \quad I_4$$

$I_5 = \text{GOTO}(I_2, id) = \phi = \text{GOTO}(I_2, integer) = \text{GOTO}(I_2, num) = \text{GOTO}(I_2, :=)$

* $I_5 = \text{GOTO}(I_3, A) = \phi = \text{GOTO}(I_3, D) = \text{GOTO}(I_3, S) = \text{GOTO}(I_3, E) =$
$= \text{GOTO}(I_3, ;) = \text{GOTO}(I_3, id)$

$I_5 = \text{GOTO}(I_3, integer) = \text{cierre}(\{ D \to id\ integer. \}) = \boxed{\{ D \to id\ integer. \}} \quad I_5$

$I_6 = \text{GOTO}(I_3, num) = \phi = \text{GOTO}(I_3, :=)$

* $I_6 = \text{GOTO}(I_4, A) = \phi$

$I_6 = \text{GOTO}(I_4, D) = \text{cierre}(\{ D \to D;D.,\ D \to D.;D \}) =$

$$= \boxed{\{ D \to D;D.,\ D \to D.;D \}} \quad I_6$$

$I_7 = \text{GOTO}(I_4, S) = \text{cierre}(\{ A \to D;S.,\ S \to S.;S \}) = \boxed{\{ A \to D;S.,\ S \to S.;S \}} \quad I_7$

$I_8 = \text{GOTO}(I_4, E) = \phi = \text{GOTO}(I_4, ;)$

$I_8 = \text{GOTO}(I_4, id) = \text{cierre}(\{ S \to id.:= E,\ D \to id.integer \}) =$

$$= \boxed{\{ S \to id.:= E,\ D \to id.integer \}} \quad I_8$$

$I_9 = \text{GOTO}(I_4, integer) = \phi = \text{GOTO}(I_4, num) = \text{GOTO}(I_4, :=)$

* $I_9 = \text{GOTO}(I_5, X) = \phi \quad \forall X$

* $I_9 = \text{GOTO}(I_6, A) = \phi = \text{GOTO}(I_6, D) = \text{GOTO}(I_6, S) = \text{GOTO}(I_6, E)$

$I_9 = GOTO(I_6, ;) = cierre(\{D \to D;.D\}) =$ $\boxed{\{D \to D;.D,\ D \to .D;D,\ D \to .id\ integer,\ D \to .\}}$ I_9

$I_{10} = GOTO(I_6, id) = \phi = GOTO(I_6, integer) = GOTO(I_6, num) = GOTO(I_6, :=)$

* $I_{10} = GOTO(I_7, A) = \phi = GOTO(I_7, D) = GOTO(I_7, S) = GOTO(I_7, E)$

$I_{10} = GOTO(I_7, ;) = cierre(\{S \to S;.S\}) =$ $\boxed{\{S \to S;.S,\ S \to .S;S,\ S \to .id := E,\ S \to .\}}$ I_{10}

$I_{11} = GOTO(I_7, id) = \phi = GOTO(I_7, integer) = GOTO(I_7, num) = GOTO(I_7, :=)$

* $I_{11} = GOTO(I_8, A) = \phi = GOTO(I_8, D) = GOTO(I_8, S) = GOTO(I_8, E) = GOTO(I_8, id)$

$I_{11} = GOTO(I_8, integer) = cierre(\{D \to id\ integer.\}) =$ $\boxed{\{D \to id\ integer.\}}$

Igual que I_5 => No añadir a C.

$I_{11} = GOTO(I_8, num) = \phi$

$I_{11} = GOTO(I_8, :=) = cierre(\{S \to id := .E\}) =$ $\boxed{\{S \to id := .E,\ E \to .id,\ E \to .num\}}$ I_{11}

* $I_{12} = GOTO(I_9, A) = \phi$

$I_{12} = GOTO(I_9, D) = cierre(\{D \to D;D.,\ D \to D.;D\}) =$

$= \boxed{\{D \to D;D.,\ D \to D.;D\}}$ Igual que I_6 => No añadir a C.

$I_{12} = GOTO(I_9, S) = \phi = GOTO(I_9, E) = GOTO(I_9, ;)$

$I_{12} = GOTO(I_9, id) = cierre(\{D \to id.integer\}) =$ $\boxed{\{D \to id.integer\}}$

Igual que I_3 => No añadir a C.

I_{12} = GOTO (I_9, integer) = ϕ = GOTO (I_9, num) = GOTO (I_9, :=)

* I_{12} = GOTO (I_{10}, A) = ϕ = GOTO (I_{10}, D)

I_{12} = GOTO (I_{10}, S) = cierre ({ S → S ; S. , S → S.; S }) =

= { S → S ; S. , S → S.; S } I_{12}

I_{13} = GOTO (I_{10}, E) = ϕ = GOTO (I_{10}, ;)

I_{13} = GOTO (I_{10}, id) = cierre ({ S → id. := E }) = { S → id. := E } I_{13}

I_{14} = GOTO (I_{10}, integer) = ϕ = GOTO (I_{10}, num) = GOTO (I_{10}, :=)

* I_{14} = GOTO (I_{11}, A) = ϕ = GOTO (I_{11}, D) = GOTO (I_{11}, S)

I_{14} = GOTO (I_{11}, E) = cierre ({ S → id := E. }) = { S → id := E. } I_{14}

I_{15} = GOTO (I_{11}, ;) = ϕ

I_{15} = GOTO (I_{11}, id) = cierre ({ E → id. }) = { E → id. } I_{15}

I_{16} = GOTO (I_{11}, integer) = ϕ

I_{16} = GOTO (I_{11}, num) = cierre ({ E → num. }) = { E → num. } I_{16}

I_{17} = GOTO (I_{11}, :=) = ϕ

* I_{17} = GOTO (I_{12}, A) = ϕ = GOTO (I_{12}, D) = GOTO (I_{12}, S) = GOTO (I_{12}, E)

I_{17} = GOTO (I_{12}, ;) = cierre ({ S → S ;. S }) = { S → S ;. S , S → . S ; S ,
S → . id := E , S → . }

Igual que I_{10} => No añadir a C.

I_{17} = GOTO (I_{12}, id) = ϕ = GOTO (I_{12}, integer) = GOTO (I_{12}, num) =
GOTO (I_{12}, :=)

* $I_{17} = GOTO(I_{13}, A) = \phi = GOTO(I_{13}, D) = GOTO(I_{13}, S) = GOTO(I_{13}, E) =$
$= GOTO(I_{13}, ;) = GOTO(I_{13}, id) = GOTO(I_{13}, integer) =$
$= GOTO(I_{13}, num)$

$I_{17} = GOTO(I_{13}, :=) = cierre(\{ S \to id := .E \}) =$ $\{ S \to id := .E,\ E \to .id,\ E \to .num \}$

Igual que I_{11} ⇒ No añadir a C.

* $I_{17} = GOTO(I_{14}, X) = \phi \quad \forall X$

* $I_{17} = GOTO(I_{15}, X) = \phi \quad \forall X$

* $I_{17} = GOTO(I_{16}, X) = \phi \quad \forall X$

Buscamos los conflictos:

Estados a comprobar → $I_0, I_4, I_6, I_7, I_9, I_{10}, I_{12}$

First (A) = { id , ; }
First (D) = { id , ; }
First (S) = { id , ; }
First (E) = { id , num }

Follow (A) = { $ }
Follow (D) = { ; }
Follow (S) = { ; $ }
Follow (E) = { ; , $ }

I_0 → ¿ id ∈ Follow(D) ? NO ✓

I_4 → ¿ id ∈ Follow (S) ? NO
¿ id ∈ Follow(D) ? NO
¿ Follow(S) ∩ Follow(D) ? Sí ⇒ Conflicto Red.- Red. Puedo reducir por $\begin{cases} A \to D;.S \\ D \to D;.D \end{cases}$

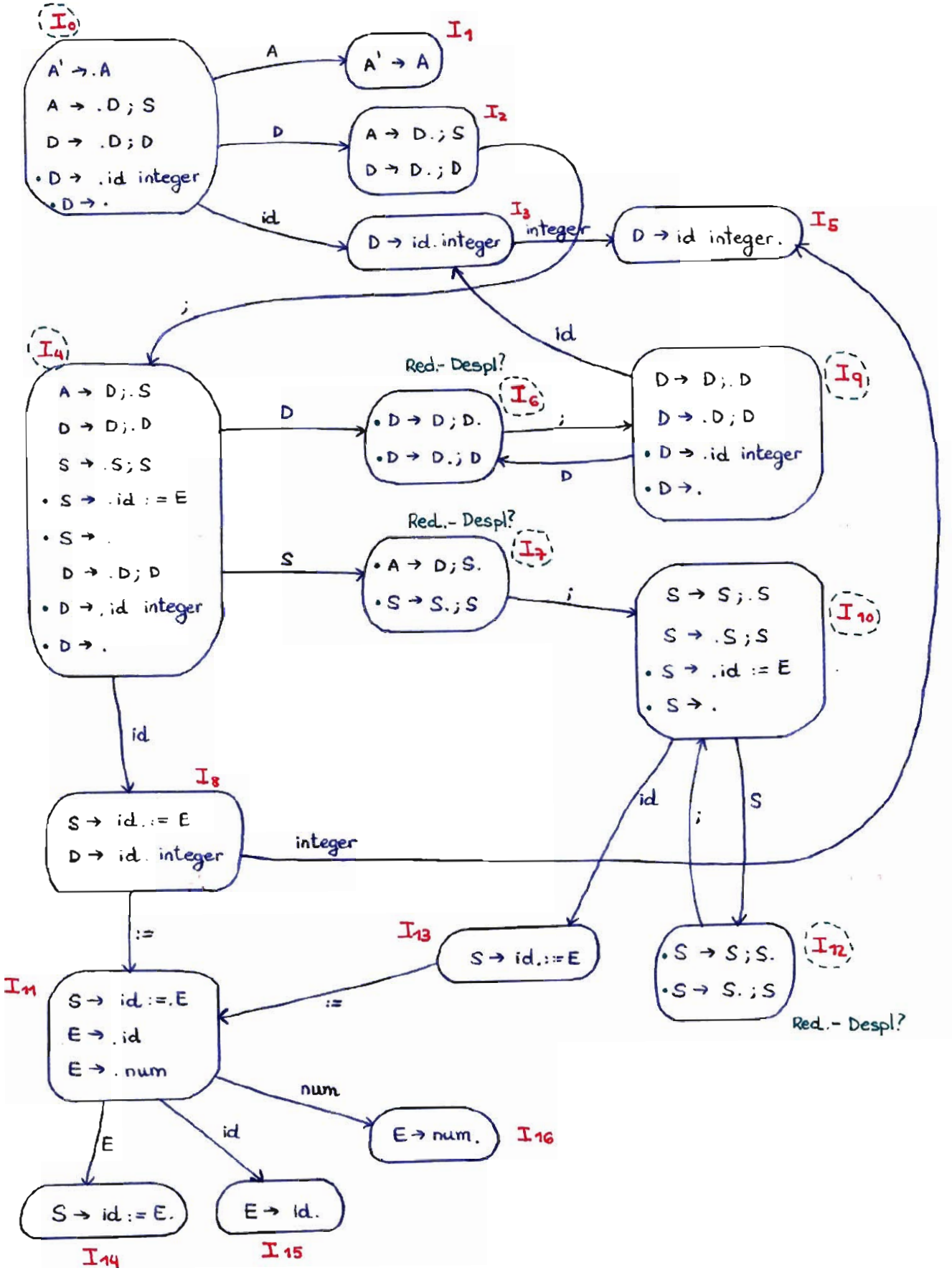
I_6 → ¿ ; ∈ Follow(D) ? Sí ⇒ Conflicto Red.- Despl.

I_7 → ¿ ; ∈ Follow (A) ? No ✓

I_9 → ¿ id ∈ Follow (D) ? No ✓

I_{10} → ¿ id ∈ Follow (S) ? NO ✓

I_{12} → ¿ ; ∈ Follow (S) ? Sí ⇒ Conflicto Red.- Despl.


I0
A' → .A
A → .D ; S
D → .D ; D
• D → .id integer
• D → .
A
I1
A' → A
D
I2
A → D.; S
D → D.; D
id
I3
D → id.integer
integer
I5
D → id integer.
;
id
I4
A → D;.S
D → D;.D
S → .S; S
• S → .id : = E
• S →
D → .D; D
• D → .id integer
• D → .
Red.- Despl?
D
I6
• D → D; D.
• D → D.; D
;
D
I9
D → D;.D
D → .D; D
• D → .id integer
• D →.
Red.- Despl?
S
I7
• A → D; S.
• S → S.; S
;
I10
S → S;.S
S → .S; S
• S → .id := E
• S →.
id
I8
S → id.:= E
D → id. integer
integer
id
;
S
:=
I13
S → id.:=E
:=
I11
S → id :=.E
E → .id
E → . num
I12
• S → S; S.
• S → S.; S
Red.- Despl?
num
E → num.
I16
E
id
S → id := E.
I14
E → Id.
I15

Ejercicio: LISP

F → (id L)
L → DA
L → DAR
A → λ
A → DA
R → rest A
D → (id e)
F' → F → Gramática aumentada.

F es el axioma (es el único símbolo no terminal que no está en el lado derecho de una producción).

Calculamos la colección canónica:

1. I_0 = cierre ({ F' → .F }) = $\boxed{\{ F' \to .F ,\ F \to .(\text{id } L) \}}$ I_0

2. C = I_0 (Inicializar)

3. j = 1

N = { F, L, D, A, R }
T = { (,), id, rest, e }
X = { F, L, D, A, R, (,), id, rest, e }

* 4. I_1 = GOTO (I_0, F) = cierre ({ F' → F. }) = $\boxed{\{ F' \to F. \}}$ I_1

I_2 = GOTO (I_0, L) = ϕ = GOTO (I_0, D) = GOTO (I_0, A) = GOTO (I_0, R)

I_2 = GOTO (I_0, ( ) = cierre ({ F → (. id L) }) = $\boxed{\{ F \to (. \text{id } L) \}}$ I_2

I_3 = GOTO (I_0,)) = ϕ = GOTO (I_0, id) = GOTO (I_0, rest) = GOTO (I_0, e)

* I_3 = GOTO (I_1, X) = ϕ ∀X

* I_3 = GOTO (I_2, F) = ϕ = GOTO (I_2, L) = GOTO (I_2, D) = GOTO (I_2, A) =
= GOTO (I_2, R) = GOTO (I_2, ( ) = GOTO (I_2,))

I_3 = GOTO (I_2, id) = cierre ({ F → (id. L) }) = $\boxed{\{ F \to (\text{id}. L),\ L \to .DA,\ L \to .DAR,\ D \to .(\text{id } e) \}}$ I_3

$I_4 = GOTO(I_2, rest) = \phi = GOTO(I_2, e)$

* $I_4 = GOTO(I_3, F) = \phi$

$I_4 = GOTO(I_3, L) = cierre(\{F \to (id\ L.)\}) = \boxed{\{F \to (id\ L.)\}}\ I_4$

$I_5 = GOTO(I_3, D) = cierre(\{L \to D.A,\ L \to D.AR\}) =$

$= \boxed{\{L \to D.A,\ L \to D.AR,\ A \to .,\ A \to .DA,\ D \to .(id\ e)\}}\ I_5$

$I_6 = GOTO(I_3, A) = \phi = GOTO(I_3, R)$

$I_6 = GOTO(I_3, () = cierre(\{D \to (.id\ e)\}) = \boxed{\{D \to (.id\ e)\}}\ I_6$

$I_7 = GOTO(I_3,)) = \phi = GOTO(I_3, id) = GOTO(I_3, rest) = GOTO(I_3, e)$

* $I_7 = GOTO(I_4, F) = \phi = GOTO(I_4, L) = GOTO(I_4, D) = GOTO(I_4, A) =$
$= GOTO(I_4, R) = GOTO(I_4, ()$

$I_7 = GOTO(I_4,)) = cierre(\{F \to (id\ L).\}) = \boxed{\{F \to (id\ L).\}}\ I_7$

$I_8 = GOTO(I_4, id) = \phi = GOTO(I_4, rest) = GOTO(I_4, e)$

* $I_8 = GOTO(I_5, F) = \phi = GOTO(I_5, L)$

$I_8 = GOTO(I_5, D) = cierre(\{A \to D.A\}) = \boxed{\{A \to D.A,\ A \to .,\ A \to .DA,\ D \to .(id\ e)\}}\ I_8$

$I_9 = GOTO(I_5, A) = cierre(\{L \to DA.,\ L \to DA.R\}) =$

$= \boxed{\{L \to DA.,\ L \to DA.R,\ R \to .rest\ A\}}\ I_9$

$I_{10} = GOTO(I_5, R) = \phi$

$I_{10} = GOTO(I_5, () = cierre(\{D \to (.id\ e)\}) = \boxed{\{D \to (.id\ e)\}}$ Igual que $I_6 \Rightarrow$ No añadir a C.

$I_{10} = GOTO(I_5,)) = \phi = GOTO(I_5, id) = GOTO(I_5, rest) = GOTO(I_5, e)$

* $I_{10} = GOTO(I_6, F) = \phi = GOTO(I_6, L) = GOTO(I_6, D) = GOTO(I_6, A) = GOTO(I_6, R) = GOTO(I_6, () = GOTO(I_6,))$

$I_{10} = GOTO(I_6, id) = cierre(\{D \to (id.e)\}) = \boxed{\{D \to (id.e)\}}$ I_{10}

$I_{11} = GOTO(I_6, rest) = \phi = GOTO(I_6, e)$

* $I_{11} = GOTO(I_7, X) = \phi \quad \forall X$

* $I_{11} = GOTO(I_8, F) = \phi = GOTO(I_8, L)$

$I_{11} = GOTO(I_8, D) = cierre(\{A \to D.A\}) = \boxed{\{A \to D.A,\ A \to .,\ A \to .DA,\ D \to .(id\ e)\}}$

Igual que I_8 => No añadir a C.

$I_{11} = GOTO(I_8, A) = cierre(\{A \to DA.\}) = \boxed{\{A \to DA.\}}$ I_{11}

$I_{12} = GOTO(I_8, R) = \phi$

$I_{12} = GOTO(I_8, () = cierre(\{D \to (.id\ e)\}) = \boxed{\{D \to (.id\ e)\}}$ Igual que I_6 => No añadir a C.

$I_{12} = GOTO(I_8,)) = \phi = GOTO(I_8, id) = GOTO(I_8, rest) = GOTO(I_8, e)$

* $I_{12} = GOTO(I_9, F) = \phi = GOTO(I_9, L) = GOTO(I_9, D) = GOTO(I_9, A)$

$I_{12} = GOTO(I_9, R) = cierre(\{L \to DAR.\}) = \boxed{\{L \to DAR.\}}$ I_{12}

$I_{13} = GOTO(I_9, () = \phi = GOTO(I_9,)) = GOTO(I_9, id)$

$I_{13} = GOTO(I_9, rest) = cierre(\{R \to rest.A\}) = \boxed{\{R \to rest.A,\ A \to .,\ A \to .DA,\ D \to .(id\ e)\}}$ I_{13}

$I_{14} = GOTO(I_9, e) = \phi$

* $I_{14} = GOTO(I_{10}, F) = \phi = GOTO(I_{10}, L) = GOTO(I_{10}, D) = GOTO(I_{10}, A) = GOTO(I_{10}, R) = GOTO(I_{10}, () = GOTO(I_{10},)) = GOTO(I_{10}, id) = GOTO(I_{10}, rest)$

I_{14} = GOTO (I_{10}, e) = cierre ({ D → (id e.)}) = $\boxed{\{ D \rightarrow (id\ e.)\}}$ I_{14}

* I_{15} = GOTO (I_{11}, X) = ϕ $\forall X$

* I_{15} = GOTO (I_{12}, X) = ϕ $\forall X$

* I_{15} = GOTO (I_{13}, F) = ϕ = GOTO (I_{13}, L)

I_{15} = GOTO (I_{13}, D) = cierre ({ A → D.A }) = $\boxed{\{ A \rightarrow D.A,\ A \rightarrow .,\ A \rightarrow .DA,\ D \rightarrow .(id\ e)\}}$

Igual que I_8 ⇒ No añadir a C.

I_{15} = GOTO (I_{13}, A) = cierre ({ R → rest A.}) = $\boxed{\{ R \rightarrow rest\ A.\}}$ I_{15}

I_{16} = GOTO (I_{13}, R) = ϕ

I_{16} = GOTO (I_{13}, ( ) = cierre({D → (. id e)}) = $\boxed{\{D \rightarrow (.id\ e)\}}$ Igual que I_6 ⇒ No añadir a C.

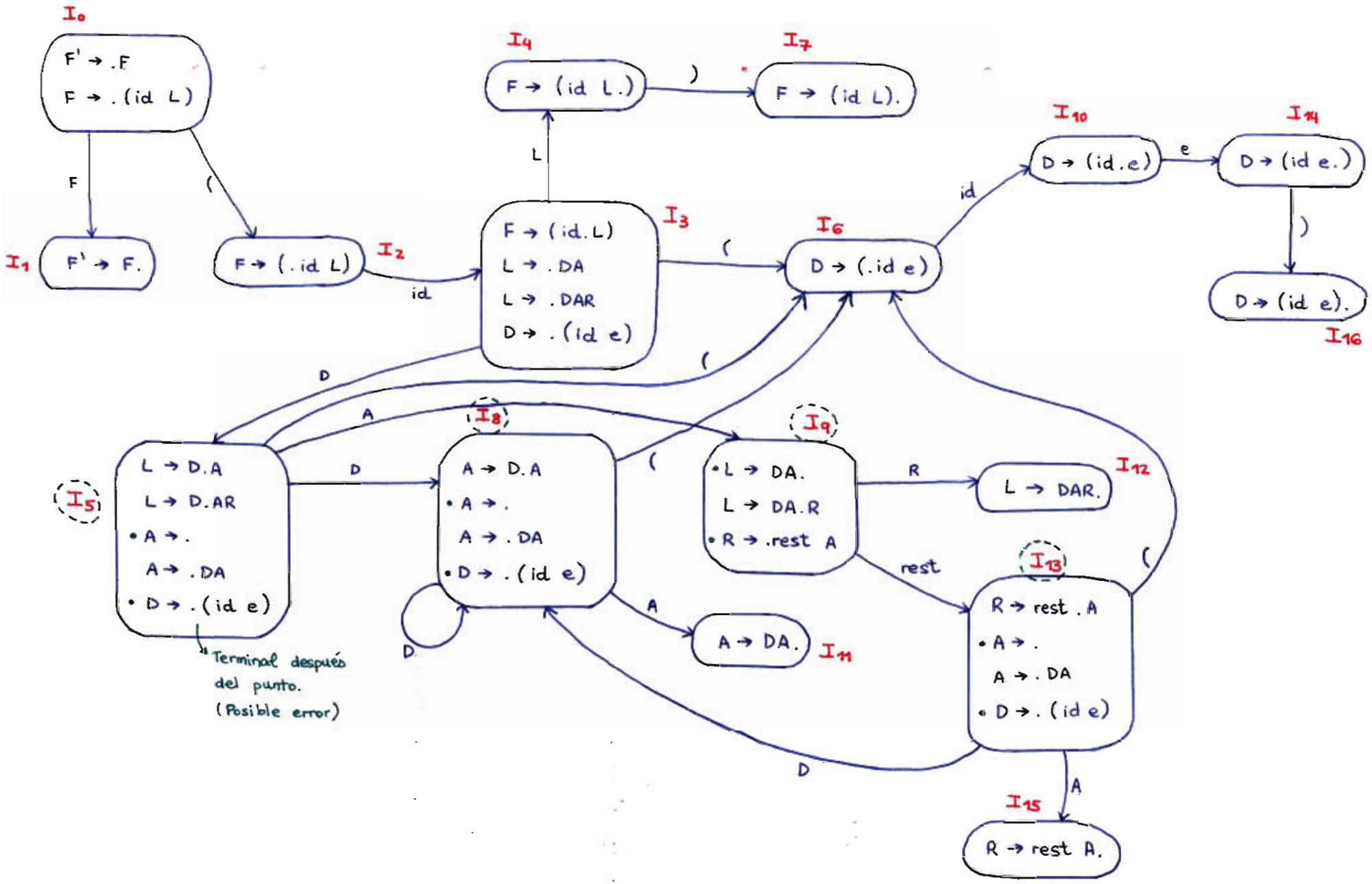
I_{16} = GOTO (I_{13},)) = ϕ = GOTO (I_{13}, id) = GOTO (I_{13}, rest) = GOTO (I_{13}, e)

* I_{16} = GOTO (I_{14}, F) = ϕ = GOTO (I_{14}, L) = GOTO (I_{14}, D) = GOTO (I_{14}, A) = = GOTO (I_{14}, R) = GOTO (I_{14}, ( )

I_{16} = GOTO (I_{14},)) = cierre ({ D → (id e).}) = $\boxed{\{ D \rightarrow (id\ e).\}}$ I_{16}

I_{17} = GOTO (I_{14}, id) = ϕ = GOTO (I_{14}, rest) = GOTO (I_{14}, e)

* I_{17} = GOTO (I_{15}, X) = ϕ $\forall X$


I0
F' → .F
F → .(id L)
F
(
I1
F' → F.
F → (.id L)
I2
id
F → (id.L)
L → .DA
L → .DAR
D → .(id e)
I3
L
I4
F → (id L.)
)
I7
F → (id L).
I6
D → (.id e)
id
I10
D → (id.e)
e
I14
D → (id e.)
)
D → (id e).
I16
D
A
I8
A → D.A
• A → .
A → .DA
• D → .(id e)
D
A
A → DA.
I11
I9
• L → DA.
L → DA.R
• R → .rest A
R
L → DAR.
I12
rest
I13
R → rest .A
• A → .
A → .DA
• D → .(id e)
D
A
I15
R → rest A.
I5
L → D.A
L → D.AR
• A → .
A → .DA
• D → .(id e)
Terminal después del punto. (Posible error)

Buscamos los conflictos:

Estados a comprobar → I_5, I_8, I_9, I_{13}

First (F) = { (}
First (L) = { (}
First (D) = { (}
First (A) = { λ, (}
First (R) = { rest }

Follow (F) = { $ }
Follow (L) = {) }
Follow (D) = { (,), rest }
Follow (A) = {), rest }
Follow (R) = {) }

I_5 → (∉ Follow (A) ✓

I_8 → (∉ Follow (A) ✓

I_9 → rest ∉ Follow (L) ✓

I_{13} → (∉ Follow (A) ✓

FIRST

Si α es cualquier cadena de símbolos gramaticales, se considera *FIRST(α)* como el conjunto de terminales que encabezan las cadenas derivadas de α. Si $\alpha \overset{*}{\Rightarrow} \lambda$, entonces λ también está en *FIRST(α)*.

Para calcular *FIRST(X)* para algún símbolo X de la gramática, se aplican las siguientes reglas hasta que no se pueda añadir nada nuevo al conjunto *FIRST*:

1. Si X es terminal, entonces *FIRST(X)* es *{X}*.

2. Si X es no terminal y existe la producción $X \rightarrow \lambda$, entonces añadir λ a *FIRST(X)*.

3. Si X es no terminal y $X \rightarrow Y_1\ Y_2 \ldots Y_k$ es una producción entonces, para todo i (con i variando desde 1 hasta k) tal que Y_1 , Y_2 , ..., Y_{i-1} sean todos no terminales y *FIRST*(Y_1), *FIRST*(Y_2), ..., *FIRST*(Y_{i-1}) contengan todos λ, se añaden todos los símbolos no nulos de *FIRST*(Y_i) a *FIRST(X)*. Finalmente, si λ está en *FIRST*(Y_j) para j = 1, 2, ..., k (o sea, en todos), entonces se añade λ a *FIRST(X)*.
   Dicho de otra forma, lo anterior significa que todos los elementos de *FIRST*(Y_1), excepto λ, pertenecen también a *FIRST(X)*. Si Y_1 no deriva λ, entonces ya ha terminado el cálculo de *FIRST(X)*, pero en caso contrario, es decir, si $Y_1 \overset{*}{\Rightarrow} \lambda$, entonces todos los elementos de *FIRST*(Y_2) excepto λ pertenecen también a *FIRST(X)*, y así sucesivamente. Finalmente, si todos los Y_i derivan λ, entonces λ se añade a *FIRST(X)*.

FOLLOW

Se define *FOLLOW(A)*, para el no terminal A, como el conjunto de terminales a que pueden aparecer inmediatamente a la derecha de A en alguna forma sentencial, es decir, el conjunto de terminales a tal que haya una derivación de la forma $S \overset{*}{\Rightarrow} \alpha A a \beta$ para algún α y β. Si A puede ser el símbolo de más a la derecha en alguna forma sentencial, entonces $ está en *FOLLOW(A)*.

Para calcular *FOLLOW(A)* para un símbolo no terminal A, se aplican las siguientes reglas hasta que no se pueda añadir nada más al conjunto *FOLLOW*.

1. $ está en *FOLLOW(S)*, siendo S el axioma de G.

2. Si existe una producción $A \rightarrow \alpha B \beta$, entonces todo lo que esté en *FIRST*(β), excepto λ, está en *FOLLOW(B)*.

3. Si existe la producción $A \rightarrow \alpha B \beta$ y *FIRST*(β) contiene λ (es decir, $\beta \overset{*}{\Rightarrow} \lambda$), o bien si existe una producción $A \rightarrow \alpha B$, entonces todo lo que esté en *FOLLOW(A)* está en *FOLLOW(B)*.

1

ALGORITMO DE ANÁLISIS SINTÁCTICO LR

ENTRADA:

- una cadena de entrada *w* con el delimitador *$* por la derecha
- una tabla de análisis LR con las funciones *ACCION* y *GOTO* para una gramática *G*

SALIDA:

- un análisis ascendente de *w*, en caso de que pertenezca a *L(G)*, o la indicación de *error* en caso contrario

MÉTODO:

Inicialmente, la pila contiene el estado inicial s_0, y la cadena de entrada *w$* está pendiente de ser analizada. El analizador ejecuta entonces el siguiente programa, del que saldrá al encontrar una acción de *aceptar* o de *error*:

```
p apunta al primer token de la cadena w$
repeat
        begin
                sea s el estado de la cima de la pila y a el símbolo apuntado por p
                if ACCION[s,a]=desp s'
                        then begin
                                meter a en la pila;
                                meter s' en la pila;
                                avanzar p al siguiente token de la entrada
                        end
                        else if ACCION[s,a]=red A→β
                                then begin
                                        sacar 2*|β| símbolos de la pila;
                                        sea s' el estado que está ahora en la cima de la
                                        pila;
                                        meter A en la pila;
                                        obtener GOTO[s',A] y meterlo en la pila
                                end
                                else if ACCION[s,a]=aceptar
                                        then return
                                        else error()
end
```

1

Previos para la construcción de la tabla de un SLR

1. GRAMÁTICA AUMENTADA

   Si G es una gramática con símbolo inicial S, la gramática aumentada G' es la que incluye el símbolo S' y la producción $S' \rightarrow S$

2. CIERRE DE UN CONJUNTO DE ÍTEMS LR(0)

   Si I es un conjunto de ítems LR(0) de una gramática G, entonces *cierre(I)* es el conjunto de ítems construido a partir de I mediante las dos siguientes reglas:

   1. Inicialmente, todos los elementos de I se añaden a *cierre(I)*

   2. Si $A \rightarrow \alpha \,.\, B\beta$ está en *cierre(I)* y $B \rightarrow \gamma$ es una producción de G, entonces el ítem $B \rightarrow \,.\, \gamma$ se añade a *cierre(I)* si todavía no está. Se sigue aplicando esta regla 2 hasta que no se puedan añadir más ítems nuevos a *cierre(I)*.

   El algoritmo, por tanto, es el siguiente:

```
function cierre(I);
begin
        J:= I;
        repeat
                for
                        cada ítem A → α . Bβ en J y cada producción B → γ de G
                        tal que B → . γ no esté ya en J
                do
                        añadir B → . γ a J
        until no se puedan añadir más elementos a J;
        return J
end
```

3. GOTO DE UN CONJUNTO DE ÍTEMS Y UN SÍMBOLO GRAMATICAL

   Se define *goto(I,X)* como el cierre del conjunto de todos los ítems $[A \rightarrow \alpha X \,.\, \beta]$ tales que $[A \rightarrow \alpha \,.\, X\,\beta]$ esté en I. Intuitivamente, si I es el conjunto de ítems válidos para algún prefijo viable γ, *goto(I,X)* es el conjunto de ítems válidos para el prefijo viable γX.

4. CONSTRUCCIÓN DE LA COLECCIÓN CANÓNICA DE ÍTEMS LR(0) PARA UNA GRAMÁTICA AUMENTADA G'

```
procedure coleccion (G');
begin
        C:= {cierre({[S'→ . S]})}
        repeat
                for
                        cada conjunto de ítems I en C y cada símbolo gramatical X tal
                        que goto(I,X) no esté vacío y no esté ya en C
                do
                        añadir goto(I,X) a C
                until no se puedan añadir más conjuntos de elementos a C
end
```

3

Construcción de la tabla de análisis de un SLR

ENTRADA:

Una gramática aumentada G'

SALIDA:

Las funciones *ACCION* y *GOTO* de la tabla de un analizador sintáctico SLR para G'

MÉTODO:

1. Construir la colección canónica de ítems LR(0) para G':

   $C = \{I_0, I_1, ..., I_n\}$

2. El estado i es el que se obtiene a partir de I_i. Las acciones para el estado i se determinan de la siguiente manera:

   a) Si $[A \rightarrow \alpha . a \beta]$ está en I_i y $goto(Ii,a)=Ij$ entonces $ACCION[i,a]$ = ***"desplazar** j"*. En este caso, a debe ser un terminal, y j es el estado que se apila tras el símbolo desplazado.

   b) Si $[A \rightarrow \alpha .]$ está en I_i entonces $ACCION[i,a]$ = ***"reducir** por $A \rightarrow \alpha$"*, y esto para todo terminal a perteneciente a $FOLLOW(A)$. En este caso, A es cualquier no terminal excepto S'.

   c) Si $[S' \rightarrow S .]$ está en I_i entonces $ACCION[i,\$]$ = ***"aceptar"***.

   Si estas reglas generan conflicto en alguna casilla, se dice que la gramática no es SLR(1). En tal caso, no es posible construir un analizador sintáctico mediante este algoritmo.

3. Las transiciones *GOTO* del estado i se construyen para todos los no terminales A mediante la regla: si $goto(I_i,A)=I_j$ entonces $GOTO[i,A] = j$.

4. Todas las casillas no definidas por las reglas 2 y 3 quedan en blanco y corresponden a los casos de ***"error"***.

5. El estado inicial del analizador es el construido a partir del conjunto de ítems que contiene a $[S' \rightarrow . S]$.

4